Journal of Software Engineering and Applications, 2014, 7, 626-638
Published Online July 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.78058

Refining Use/Misuse/Mitigation Use Cases for Security Requirements

Joshua J. Pauli

College of Business & Information Systems, Dakota State University, Madison, USA
Email: Josh.Pauli@dsu.edu

Received 24 April 2014; revised 23 May 2014; accepted 20 June 2014

Abstract

We investigate security at the same time as the functional requirements by refining and integrating use, misuse, and mitigation use cases. Security requirements rely on the interactions among normal system execution (use cases), attacks (misuse cases), and necessary security strategies (mitigation use cases), but previous approaches only use a high-level of abstraction. We use refinement to uncover details of each case and the relationships among them before integrating them. We identify and model "includes" and "extends" relationships within each refined case type, and use a condition-driven process that maintains these relationships as refinement continues. We then systematically identify and model "threatens" and "mitigates" relationships to integrate the cases at a detailed level.

Keywords

Use Case, Misuse Case, Mitigation Use Case, Requirements, Security Engineering

1. Introduction

An issue with software security is the accuracy and consistency of the requirements. Software security is often investigated after other development stages are complete, thus overlooked or not investigated with the resources that it needs to be successful [1] [2]. Many problems that arise during development can be prevented with the proper requirements in place from the project inception. Getting the system's requirements correct initially costs 50 - 200 times less than correcting code later [3]. Likewise, every hour that is spent on reviewing the requirements saves 33 hours of maintenance on average during the lifetime of the system [4]. Monetary cost is too often used as the ultimate indicator as to what exactly a software system can include; security has been one of the properties of the system that has been forsaken.

Security is needed in the requirements phase so that work that takes place later in the development process

How to cite this paper: Pauli, J.J. (2014) Refining Use/Misuse/Mitigation Use Cases for Security Requirements. *Journal of Software Engineering and Applications*, **7**, 626-638. http://dx.doi.org/10.4236/jsea.2014.78058

can be based on functional and security requirements. The most popular approach is misuse case modeling [5]-[10]. Integrated models of use/misuse/mitigation use cases are modeled strictly at a high-level of abstraction where no details of the system are known. We use refinement to investigate the details of these case interactions. Refinement is used in requirements specifications to represent simple, unified, units of work [11]. Criterion in which the refinement occurs is of utmost importance because *ad hoc* refinement is not good enough to base development decisions on [12].

Our methodology combines misuse case modeling and requirements refinement. Our approach includes three phases. In the first phase, we identify the refined cases from high-level textual descriptions and model "includes" and "extends" relationships. In the second phase, we refine each case type with emphasis on maintenance of "includes" and "extends" relationships. In the third phase, the "threatens" and "mitigates" relationships are identified and modeled among the use/misuse/mitigation use cases.

This paper is structured where Section 2 covers related work. Section 3 covers our approach to refining each case type. Section 4 covers how we maintain the "includes" and "extends" relationships. Section 5 covers the process of identifying, modeling, and maintaining the "threatens" and "mitigates" relationships. We conclude in Section 6.

2. Related Work

Misuse cases have roots in traditional use cases to model possible system attackers. Once fully modeled with mitigation use cases, use cases and misuse cases play a game of "cat and mouse" within the model where misuse cases threaten use cases and mitigation use cases prevent misuse cases. A misuse case is simply a use case from the point of view of an actor hostile to the system [7]. Misuse cases seem naturally to lead to non-functional requirements such as safety and security, whereas use cases essentially describe system behaviors in functional terms [6]. Using both use and misuse cases to model scenarios in the system improves security by helping to mitigate threats [6]. Misuse cases are generally depicted by black ovals, while normal use cases are depicted by white ovals. Each misuse case "threatens" one or more use cases with failure [6]. Misuse cases are highly effective ways of analyzing security threats, but are inappropriate for the analysis and specification of security requirements; these security requirements should come from security use cases or mitigation use case [6]. All cases (use, misuse, mitigation use) make use of textual descriptions to convey further details about the misuse. The common fields that would be included in such a description would include name, summary (describing the interaction performed by the misuse case), and basic path (normal path of action taken, ending with success for the misuser and thus failure for the system and its owners) [13]. Other fields that may be part of the textual description for a misuse case might include preconditions, assumptions, worst case threat, capture guarantee, and the scope of the misuse [14].

Use case refinement has been successfully applied previously in varying areas of software development; namely user interface design [15]. Refinement is certain if the system is of moderate to large size, but the criteria and engineering process used in these efforts are important because ad-hoc efforts are not good enough [12]. Previous efforts of use case refinement did not follow a set methodology, but instead relied on the analyst to know what pieces made up a use case. Abstractions at one layer can be related to the abstractions at other layers in a variety of ways, thus simplifying the abstractions at each layer rather than collapsing different abstractions into a more complex single layer [16]. Refining to a level of being able to solve the problem is the goal, but there is no current way to account for non-functional requirements [12]. Using refinement to investigate the details of use cases has several major advantages. It leads to a smaller, simpler use case model in which each part is comparatively simple and easy to understand in itself and in relation to a small number of other use cases [15]. The total model is simplified through reuse because common elements are factored out into separate use cases rather than repeated numerous places and variations [15]. Each piece of the problem can be solved once and the solution can be recycled wherever the corresponding use case is referred in the model where the collection of use cases is interrelated by inclusion, specialization, and extension [15]. The development process entails various temporal decompositions, such as abstraction refinement, in which the representation of a particular function, module, layer, or system interface undergoes successively increased specificity [16]. This is the type of refinement used in our approach. In use case-centered development, the capturing of requirements is based primarily on core requirements which are discharged by refinement and allocated across many subsystems [17]. Once refined, the requirements must be treated equally as they all face the same scrutiny when dealing with traceability

[17]. The logical refinement of a system is a coherent set of UML subsystems that collaborate to provide the desired behavior, however, maintainability of the models is a concern [18].

3. Refining Cases

Previous refinement efforts do not have explicit steps for guidance, thus these ad-hoc approaches do not produce accurate or consistent refined cases. Our approach is made up of exact steps and conditions that produce details concerning the interplay among uses, misuses, and mitigations. This section introduces refinement concerned with one case type (use, misuse, or mitigation) alone and what happens as they are refined as introduced in **Figure 1**.

In **Figure 1**, the high-level case ("A") is refined into three separate cases ("A_1", "A_2", and "A_3") where "A" may be a use, misuse, or mitigation use case. As the high-level case is refined, "includes" and "extends" relationships between the refined cases must be identified and modeled. We use the textual descriptions of each case to drive the refinement where each step of a case is a candidate to become a case in the next level of abstraction. Our overall refinement process can be summarized as:

1) Identify candidate cases from the high-level textual description.
2) Create initial textual descriptions for each refined case.
3) Identify if any "includes" relationship exists and how to model it for refined cases.
4) Identify if any "extends" relationship exists and how to model it for refined cases.
5) Identify and model appropriate actor assignments for each refined case.

The details of using the "includes" relationship will be introduced in Section 3.1 while the details of the "extends" relationship will be introduced in Section 3.2. Actor assignments will be introduced in Section 3.3.

3.1. Includes Relationship

"Includes" relationships should only be used after a textual description has been written for each use case [19]. We developed a set of steps that detail exactly how the "includes" relationships are identified and modeled for the refined cases. The main goal of these steps is to revise the initial textual descriptions to accurately reflect the presence of any "includes" relationships between refined cases. Once the textual descriptions have been accurately revised it is then possible to model these relationships in a use case model. Our steps to manage the "includes" relationship are listed below.

1) Check the refined cases for any "includes" relationship by using custom conditions. This check involves inspecting the preconditions and steps of the refined cases.
2) Identify the refined case that accounts for the common or shared behaviors from the textual descriptions of the refined cases. This is the "included" case.
3) Revise steps in the textual descriptions of the refined cases to reference the "included" case.
4) Model the refined cases per the updated textual descriptions.

The conditions we use to identify "includes" relationships are listed below.

1) If two or more cases share steps that are already accounted for by another refined case, then an "includes" relationship is needed among those refined cases. For example, in **Figure 1** if the "A_2" and "A_3" cases have common steps then these common steps would have to be modeled by "A_1" for an "includes" relationship to be present for the refined cases. The two separate "includes" relationships would originate at "A_2" and "A_3" and both target "A_1".

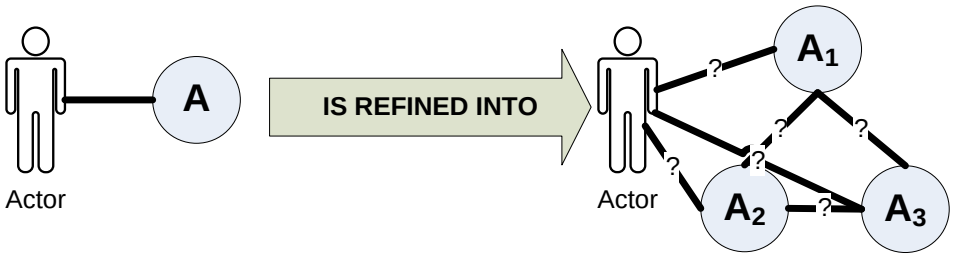

Figure 1. Refinement of individual case type.

2) If two or more cases have the same precondition and the precondition is a post-condition of a separate refined case, then an "includes" relationship is needed among those refined cases. For example, in **Figure 1** if the "A_2" and "A_3" cases have the same preconditions and the post-condition from "A_1" is the same, then "includes" relationships are needed. The two separate "includes" relationships would originate at "A_2" and "A_3" and both target "A_1".
3) If two or more cases have the same post-condition and the post-condition is accounted for by a separate refined case, then an "includes" relationship is needed among those refined cases. For example, in **Figure 1** if the "A_2" and "A_3" cases have the same post-conditions and that behavior is accounted by "A_1", then "includes" relationships are needed. The two separate "includes" relationships would originate at "A_2" and "A_3" and both target "A_1".

Once a condition has been met, the second step is to group these common behaviors into "included" cases to be referenced by the other cases. The third step mandates that the textual description for each involved case is revised to reflect the "included" case. Once the textual descriptions have been revised with the "includes" relationship, a model can be created per the newly revised textual descriptions; this is the fourth and final step in managing the "includes" relationship. The PIS "U-1" use case is used as an example in this section. **Table 1** introduces the textual description of the "U-1: Complete Administrative Tasks" use case textual description.

Without proper investigation the refined model would include six separate use cases and no identified relationships among them. Because of space concerns, the textual descriptions are not included. Refer to [20]-[24] for more details. The second condition is true because there are two or more cases that have the same precondition which is a post-condition for a separate refined case. "U-1.4", "U-1.5", and "U-1.6" all have the same precondition of "PIS user started the application" which is the post-condition of "U-1.3: Start System Application". The second step in managing the "includes" relationship is to identify which refined case accounts for this shared behavior. For this example it is "U-1.3". The "U-1.4", "U-1.5", and "U-1.6" use cases need to reference this case as part of their execution. The third step in managing the "includes" relationship is to revise the textual description of each refined case that is part of the relationship with this "included" use case. The fourth step in managing the "includes" relationship is to model these refined cases per the updated textual descriptions as introduced in **Figure 2**. This model only includes cases that are part of the "includes" relationship; other use cases are not included in this model for readability.

We use this same process for modeling "includes" relationships among refined misuse and mitigation use cases.

Table 1. Textual description for the "U-1: complete administrative tasks" use case.

Use Case: "U-1: Complete Administrative Tasks"
Precondition
User has proper permissions
Postcondition
User successfully completes the task(s)
Paths (steps)
1. Payroll Staff member logs onto the system
2. System authenticates the Payroll Staff to the system
3. Payroll Staff member starts system application
4. Payroll Staff member requests information from the system
5. Payroll Staff member enters information into the system
6. Payroll Staff member retrieves information from the system
Special Requirements: A guest account will have to be assigned permissions each time it is used for individual circumstances
Assumptions: All PIS users will be in good standing with correct permissions
Abstraction Level: High Level

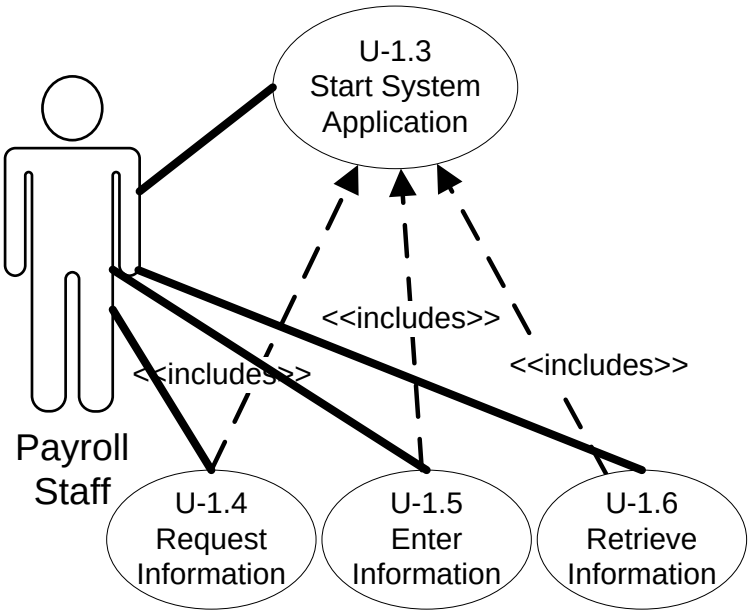

Figure 2. Refined "U-1: complete administrative tasks" use case after "includes" relationship process.

3.2. Extends Relationship

Use case modeling uses the "extends" relationship for optional behaviors or adding functionality to a case [19]. The "extends" relationship is also used to model variants in behavior [25]. We use the "extends" to model when one refined cases provides additional behavior to another refined cases, or when there are alternative executions among two or more refined cases. The steps for the "extends" relationship are similar to the "includes" relationship process.

1) Check the higher level case for an "extends" relationship by using conditions.
2) Identify the refined cases that account for the optional or additional behaviors as the originators of the "extends" relationships.
3) Identify the refined case that is the target of the "extends" relationship as the last high-level step that executes before the optional or additional behaviors execute.
4) Revise steps in the textual description of the targeted cases to reference the "extended" case.
5) Model the refined cases per the updated textual description from the targeted case.

The two conditions we use to identify "extends" relationships are listed below.

1) If a high-level case has optional behaviors the in steps then an "extends" relationship is needed. In **Figure 1**, if the "A" case has two steps that contain optional behaviors, then these two optional behaviors would be modeled by "A_2" and "A_3" after "A" was refined. The two separate "extends" relationships would originate at "A_2" and "A_3" and both target "A_1".
2) If a refined case should be accessed by many other refined cases for additional behaviors then an "extends" relationship is needed. In **Figure 3**, if the "A_2" and "A_3" cases are additional behaviors that "A_1" wants to execute, then "extends" relationships are needed for these refined cases. The two separate "extends" relationships would originate at "A_2" and "A_3" and both target "A_1".

There will one originating case when the "extends" relationship is used for additional behaviors and two or more originating cases when used for optional behaviors. To illustrate the "extends" process, we use the "S-1" mitigation use case as introduced in **Table 2**. When the steps are investigated with the conditions it is realized that a relationship is needed.

The first condition is true because the second and third steps are optional behaviors. Depending on the request type, it will be either allowed as anonymous or be allowed as valid. We identify the cases that account for the optional or additional behaviors (originating cases) as "S-1.2" and "S-1.3". We identify which refined case will be the target of the "extends" relationship as the last high-level step that executes before the optional or addi-

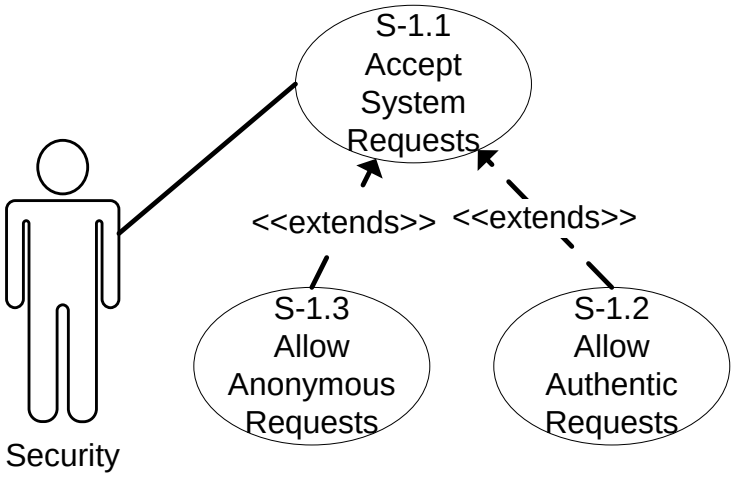

Figure 3. Intermediate level "S-1: throttle all requests" mitigation use case.

Table 2. Textual description for the "S-1: throttle all requests" mitigation use case.

Mitigation Use Case: "S-1: Throttle All Requests"
Precondition
1. Anonymous request is made to the system
2. Validated request is made to the system
Postcondition
1. DoS Attack requests have been throttled
Paths (steps)
1. Request is determined if request is anonymous or valid
2. Authentic requests are allowed to pass undeterred
3. A small number of anonymous requests are allowed to pass undeterred
Abstraction Level: High

tional behaviors execute ("S-1.1"). Finally, we revise the textual description of the targeted case to account for these originating cases. As soon as the system accepts a request from a PIS user, execution is passed to one of the cases. All valid requests will execute "S-1.2" and have resource limits enforced. All anonymous requests will execute "S-1.3" and have the IP address listened to for possible attacks. The final "S-1" model is introduced in **Figure 3**.

3.3. Actor Assignments

We developed a way to identify the appropriate actor assignments based on the information documented in the textual descriptions of each refined case as listed below.

1) List the identifier and step from the high-level cases that account for each refined case.
2) Check to see if the human actor executes each step in the high-level cases. If the human actor executes the step, then an actor assignment is needed for the corresponding refined cases.
3) If the human actor does not execute the step, then the case is considered independent from the human actor. For these independent actors the system actor assignment is needed.

We assign actors after the "includes" and "extends" relationships have been modeled, so that the textual descriptions are finalized after being revised during relationship identification.

For mitigation use cases it is not easy to identify which mitigation use case is completed by the "security" ac-

tor because the "security" actor is not an actual person. We simply assign the actor to the initial mitigation use case and allow execution to filter down to the remaining cases. This is the only consistent way to assign actors to cases that are executed by an implemented system feature and not by a human actor.

4. Ensuring Consistent Relationships

The "includes" and "extends" relationship and actor assignments must be maintained as refinement continues and additional cases are created. In **Figure 4**, "R" depicts the "includes" or "extends" relationship that was modeled as part of the initial refinement. "R" persists between the refined "A_1" and "A_2" case subsets. A case subset is the collection of refined cases from a higher level model such as the "A_1" subset made up of two cases ("$A_{1.1}$" and "$A_{1.2}$"). We must model which of the "A_2" cases originates "R" and which of the "A_1" cases is targeted by "R". This identification and modeling are further complicated by the possible presence of relationships ("R_1" and "R_2") in the case subsets. "R_1" and "R_2" may be "includes", "extends", or no relationship. The originating cases and targeted cases for "R" are identified depending on the values of "R_1" and "R_2".

We developed a condition-driven approach to identify the originating cases and targeted cases for "R". These conditions are the same when "R" represents an "includes" or "extends" relationship. These conditions are also the same when working with any of the three case types. Our conditions used to maintain the initial "includes" and "extends" relationships are listed below. Conditions 1 - 3 deal with the targeted cases and 4 - 6 deal with the originating cases of "R".

1) IF "R_1" = "includes" THEN "R" target = the originators of "R_1".
   This is because "$A_{1.2}$" has to execute because it initiates the relationship. "$A_{1.1}$" is not an originator because it is shared behavior of "$A_{1.2}$" and will execute when "$A_{1.2}$" executes.
2) IF "R_1" = "extends" THEN "R" target = the target of "R_1".
   This is because "$A_{1.1}$" has to execute because it is the first case to execute with additional behaviors attached. "$A_{1.2}$" is not a target because it is optional or additional behaviors of "$A_{1.1}$".
3) IF "R_1" = NONE THEN "R" target = the first "A_1" case to execute based on case identifier.
   This is because "R" targets the first case to execute when the cases are independent.
4) IF "R_2" = "includes" THEN "R" originator = the originators of "R_2".
   This is because "$A_{2.2}$" has to execute because it initiates the relationship. "$A_{2.1}$" is not an originator because it is shared behavior of "$A_{2.2}$" and will execute when "$A_{2.2}$" executes.
5) IF "R_2" = "extends" THEN "R" originator = the target of "R_2".
   This is because "$A_{2.1}$" has to execute because it is the first case to execute with additional behaviors attached. "$A_{2.2}$" is not a target because it is optional or additional behaviors of "$A_{2.1}$".
6) IF "R_2" = NONE THEN "R" originator = the first "A_2" case to execute based on identifier.
   This is because "R" originates from the first case to execute.

These conditions are mutually exclusive and collectively exhaustive in that only one of each condition (1 - 3 and 4 - 6) can be true and they account for every possibility.

Applying the Conditions for Both Relationship Types

The question remains, what cases in the "A_2" subset "includes" or "extends" what cases in the "A_1" subset? We use part of the "U-1" use case as an example to illustrate our conditions for identifying the targeted cases and

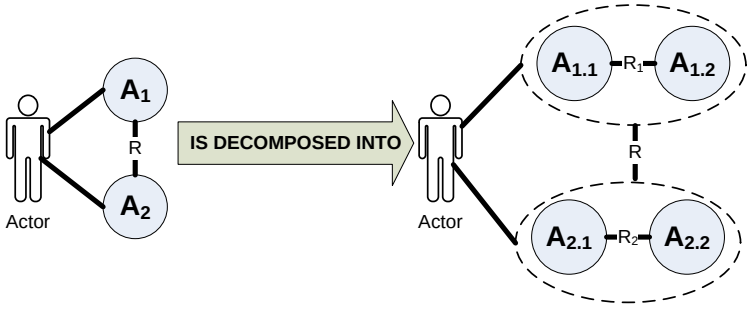

Figure 4. Generic depiction of continued refinement.

originating cases where "U-1.4", "U-1.5", and "U-1.6" all include "U-1.3". Further work is needed to identify exactly what subset case takes part in the relationship. **Figure 5** introduces what is known after the modeling of the "includes" relationship.

"U-1.3" only has one refined use case ("U-1.3.1") so there is obviously no relationship within this case subset. This meets condition #3 which states that when there is no "R_1" that "R" targets the first case to execute, so every corresponding use case subset will target "U-1.3.1" with the "includes" relationships. The "U-1.4" subset has no relationships among its refined cases; it meets condition #6, so the "includes" relationship originates from the first originating case that executes. "U-1.4.1" will originate the "includes" relationship with "U-1.3.1". The "U-1.5" subset has no relationships, so it also meets condition #6 in which the first executing case will originate the "includes" relationship. "U-1.5.1" will originate the "includes" relationship with"U-1.3.1". The "U-1.6" subset has an "extends" relationship among its refined cases, so it meets condition #5 where an "extends" relationship is present among the originating case subset. The originating cases for the "includes" relationship with the "U-1.3.1" case is the target of the "extends" relationship within the case subset. "U-1.6.1" case originates the relationship with "U-1.3.1".

Figure 6 shows which of the refined use cases originate the "includes" relationship with the targeted case ("U-1.3.1") as driven by the conditions we developed.

The model in **Figure 6** is accurate through two iterations of refinement as described by the following statements about the refined cases used in this example.

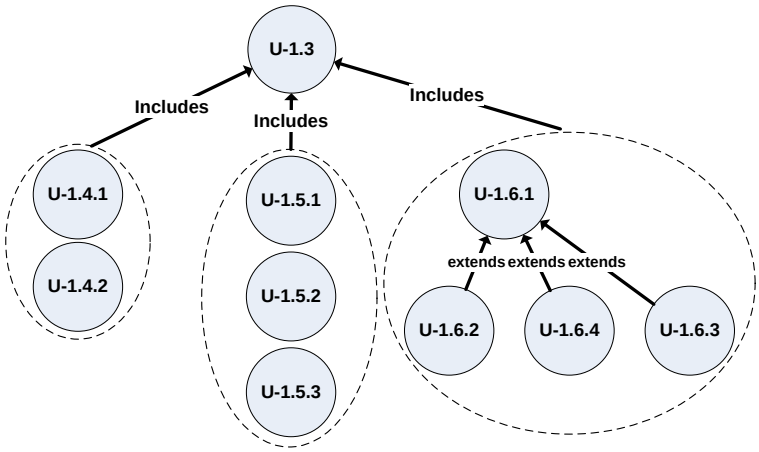

Figure 5. Refined "included" use case model.

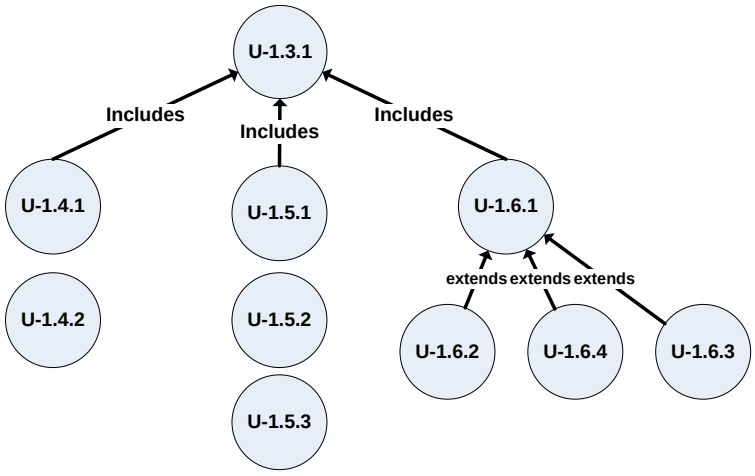

Figure 6. Final Refined "Included" Use Case Model for "U-1.3", "U-1.4", "U-1.5", and "U-1.6".

1) Request parameters must be entered before information is retrieved. Therefore, it is accurate that "U-1.4.1" comes before "U-1.4.2".
2) Hours must be entered before assigning withholdings and deductions which must happen before deductions and pay are calculated. Therefore, it is accurate that all three of the use cases execute in order. "U-1.5.1" comes before "U-1.5.2" which comes before "U-1.5.3".
3) Personal information, pay rate, W-2 information, and deductions can all be retrieved without consideration of the others. Therefore, it is accurate that the "extends" relationship drives the identification of the originating case. The target of the "extends" relationship ("U-1.6.1") originates the "includes" relationship with the "U-1.3.1" use case.

As shown in Section 3, we use textual descriptions to identify which cases need an actor assigned. If the step indicates that the human actor executes the use case, then that assignment is made. If the human actor is not present, a system actor is needed as introduced in **Figure 7**. A system actor must be modeled to account for the independent cases ("U-1.4.2" and "U-1.5.3") that do not require human intervention for execution.

Assigning actors for misuse and mitigation use cases is similar. We use "attack timing" and "mitigation timing" for the system actor for misuse and mitigation use cases because there is no system in which misuses and mitigations executed.

5. Modeling Integration Relationships

"Threatens" and "mitigates" relationships integrate functional and security requirements in our approach. When these integrated models are refined, the relationships must be modeled in an accurate and consistent manner. **Figure 8** introduces the relationship between cases where "A" and "B" are different case types that are executed by different actors. If "Actor-1" executes a use case then "Actor-2" executes a misuse cases and "R" would be a "threatens" relationship. If "Actor-1" executes a misuse case then "Actor-2" executes a mitigation use case and

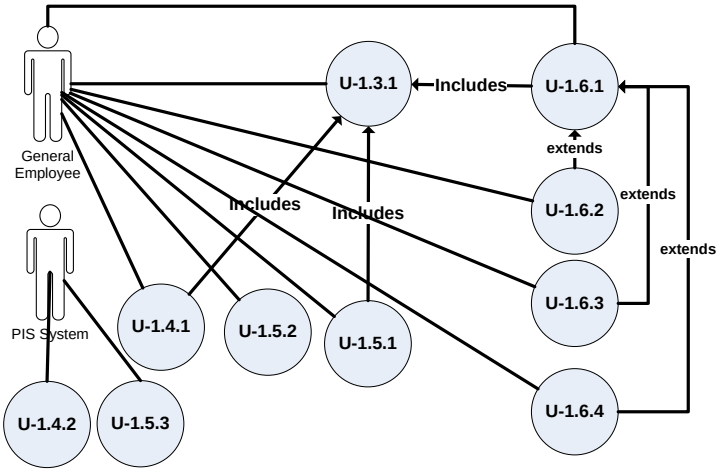

Figure 7. Refined "U-1.3", "U-1.4", "U-1.5", and "U-1.6" use cases with actor assignment.

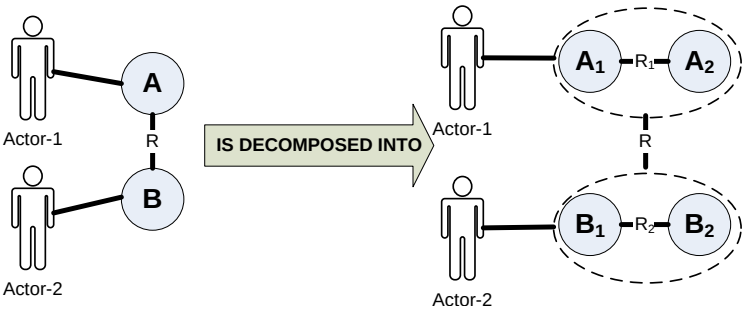

Figure 8. Relationships between different case types.

"R" would be a "mitigates" relationship. "R" is present at the highest level of abstraction as a "threatens" or "mitigates" relationship and now needs to be maintained as refinement continues.

We have three priorities related to the modeling of "threatens" and "mitigates" relationships.

1) The modeling of originating and targeted cases for "R" between case subsets.
2) The revision of these cases when there is "includes" and/or "extends" relationships present within the case subsets. This revision uses the same rules as the condition-driven approach introduced in the previous section.
3) The illustration of "threatens" and "mitigates" relationships at different levels of abstraction.

Section 5.1 introduces the process of modeling "threatens" relationships, while Section 5.2 introduces the process of modeling "mitigates" relationships

5.1. Threatens Relationship

We must properly manage this relationship to ensure the correct refined misuse cases are threatening the correct refined use cases as introduced in the following numbered list.

1) Identify threats for each refined use case from the steps in the high-level textual descriptions.
2) Identify threats produced by each misuse from the steps in the high-level textual descriptions.
3) Map each misuse case to any use case that is threatened by it. This models the refined use case as the target of the "threatens" and the exact refined misuse case as the originator.
4) Use the conditions from Section 4 to ensure mapping of "threatens" relationship between the misuse case and the use case subsets when "includes" and "extends" are present.
5) Collapse and expand the model to show "threatens" at varying levels of abstraction.

Refinement uncovered the details of each case type independently of other case types; now these refined cases are integrated together with this process. We use "U-1" and "M-2" as an example of using this process. At a high-level "M-2" threatens "U-1", but it is unknown exactly which of the refined cases are the originator(s) and target(s). The obvious question is does "M-2" really threaten each of the six use cases that make up "U-1"? To answer this, "M-2" must be investigated for the exact threats that it produces. The steps of the textual description outlines the exact threat that each refined misuse case produces as introduced in **Table 3**.

Next, we map the misuse cases to the use cases by comparing these threats (**Table 3**) to the known threats from the "U-1" textual description. If a match is made then a "threatens" relationship is present between the two cases. For example, the "M-2.1" misuse case produces the threat "hacking the system in search of login information". Both "U-1.1" and "U-1.2" are threatened by "M-2.1". When checking is complete, an integrated model shows exactly which misuse cases threaten exactly which use cases. Revisions may be needed based on the presence of any "includes" or "extends" as first introduced in Section 4. The fourth step of managing the "threatens" relationship is to use these conditions to revise the "threatens" relationships accurately. A final use case model is created after any necessary revisions are made to the "threatens" relationships as introduced in **Figure 9**.

It may be advantageous to have the ability to collapse any one part of the model to aid in understandability. This must be done only after the relationship has been accurately mapped.

5.2. Mitigates Relationship

We must also manage the "mitigates" relationship to ensure the correct mitigation use case is mitigating the correct misuse case as introduced in the following numbered list.

1) Identify the mitigations produced by each refined mitigation use case. This extracts the mitigations from each case so that these entries can be used in modeling "mitigates".

Table 3. Refined "M-2" misuse cases and produced threats.

Misuse Case	Produced Threat
M-2.1: Compromise Router	Attacker hacks the system in search of login information
M-2.2: Guess Logon Info	Attacker uses this information to login to the system as a valid user
M-2.3: View Payroll Information	Attacker view information in the system

2) Map each mitigation to applicable misuse cases based on the "mitigated by" column as the target of the "mitigates" relationship.
3) Use the conditions to ensure mapping of "mitigates" between mitigation use case and misuse case subsets when there are "includes" and "extends" present in the case subsets.
4) Collapse and expand the model to show "mitigates" at varying levels of abstraction.

This process provides the detailed interactions between mitigation use and misuse cases. The necessary security measures will be identified and modeled in this process. We continue with "M-2" as an example. "S-2" mitigates part of "M-2", but it is unknown exactly which parts of "S-2" actually prevent "M-2". Mitigations are harvested from steps in the text description of the mitigation use case. The misuse case provides what is needed to prevent the threat; this allows for the mapping of "mitigates". "M-2.1" and "M-2.2" are not mitigated by "S-2". The other mitigation use cases would need to be checked to see which actually prevents these misuse cases.

A check for "includes" or "extends" present within the case subsets is now performed. We use the same conditions to check for accuracy of the "mitigates" relationships and make revisions to the originating or targeted cases of the relationship. In this example, there is an "extends" relationship originating at "S-2.2" and targets "S-2.1". In terms of the "mitigates" relationship, these cases are the originating cases while "M-2.3" is the target. Referring to **Figure 8**, "R" is "mitigates", "R_2" is "extends" between "S-2.2" and "S-2.1"; "B_1" is "S-2.1" and "B_2" is "S-2.2". Therefore, "mitigates" would originate only from "S-2.1" because it is the target of the "extends" relationship within the case subset. A final use case model is created after any necessary revisions are made to "mitigates" as introduced in **Figure 10**.

Once "threatens" and "mitigates" have been revised, the three case types can be integrated into one model as introduced in **Figure 11**.

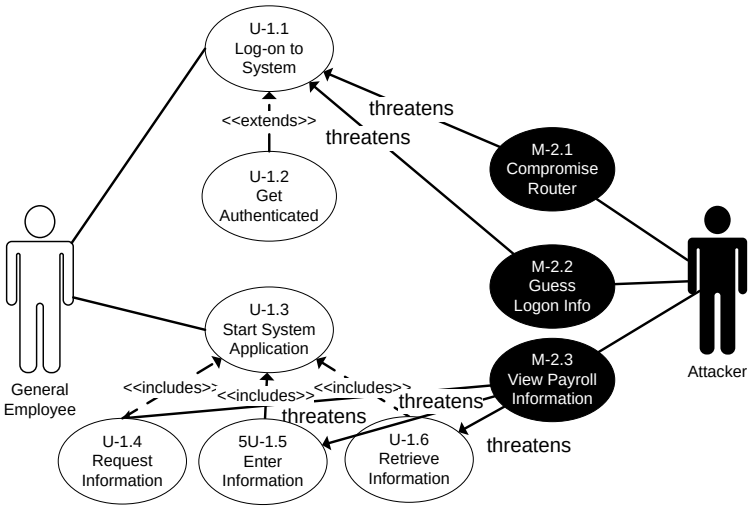

Figure 9. Use case model with "threatens" relationships.

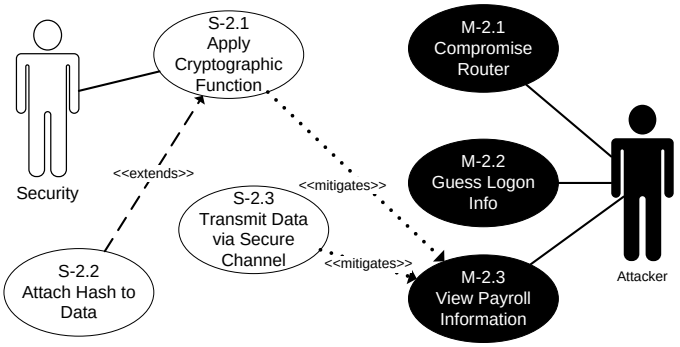

Figure 10. Final use case model with "mitigates" relationship.

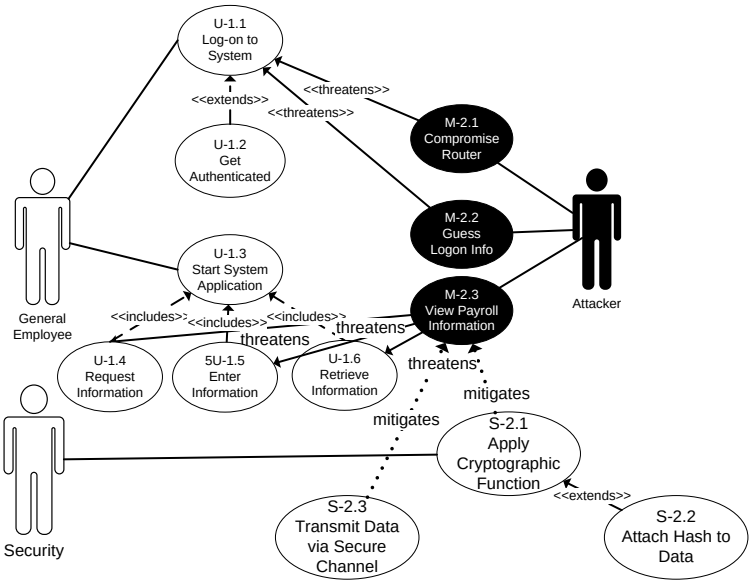

Figure 11. Final integrated use case model with use, misuse, and mitigation use cases.

This model shows exactly what use cases are threatened and how it will be mitigated. Our goal is to createrefined models that reflect the interplay of these case types at a more detailed level than the high-level use/misuse/mitigation use case models previously used.

6. Conclusions

We refine and integrate use, misuse, and mitigation use cases in the requirements phase for the benefit of security requirements by completing three specific phases. First, we refine each case type independently of other case types. Second, we maintain the accuracy and consistency of the cases as refinement continues. Third, we integrate the differing case types into a single model to investigate the detailed interactions among the differing case types.

Our refinement process is driven by the textual descriptions of each case and includes extensive work on modeling “includes” and “extends” accurately and consistently. We created a decision making process which includes steps and conditions to identify and model these relationships as part of the initial refinement of each case. We follow conditions that make an “includes” or “extends” relationship likely among the newly refined cases and apply our steps and conditions to ensure the appropriate relationships among the refined cases. Integration is the main reason for refining; we are not trying to create a functional decomposition of use cases.

The main limitation of our approach is that it is completed without any assistance from automated tools. This causes the amount of time required to appropriately use the approach to be long. It is not a trivial task to identify cases, identify and model relationships within each case type, and accurately identify and model the integrating relationships between the case types. Automated tools not only would help with the time factor of using our approach, but also would eliminate the possibility of human errors. Because of the time it takes to use our approach from start to finish, the overall size of the system is a concern. Our approach is best suited for small information systems that have a need for security and will be executed primarily by human users. Large systems would take a long time to follow our steps in identifying all of the relationships.

References

[1] Devanbu, P. and Stubblebine, S. (2000) Software Engineering for Security: A Roadmap. *Proceedings of the Conference on The Future of Software Engineering*, 227-239.

[2] Ghosh, A., Howell, C. and Whittaker, J. (2002) Building Software Securely from the Ground Up. *IEEE Software*, **19**, 14-16. http://dx.doi.org/10.1109/MS.2002.976936

[3] McGraw, G. (2004) Software Security. *IEEE Security & Privacy*, **1**, 32-35.

[4] Anton, A. and Potts, C. (1998) The Use of Goals to Surface Requirements for Evolving Systems. *Proceedings of the* 20*th International Conference on Software Engineering*, Kyoto, 19-25 April 1998, 157-166. http://dx.doi.org/10.1109/ICSE.1998.671112

[5] Alexander, I. (2002) Initial Industrial Experience of Misuse Cases in Tradeoff Analysis. *Proceedings of the* 10*th Anniversary IEEE Joint International Conference on Requirements Engineering*, Essen, 9-13 September 2002, 61-68. http://dx.doi.org/10.1109/ICRE.2002.1048506

[6] Alexander, I. (2003) Misuse Cases Help to Elicit Non-Functional Requirements. *Computing & Control Engineering Journal*, **14**, 40-45. http://dx.doi.org/10.1049/cce:20030108

[7] Alexander, I. (2003) Misuse Cases: Use Cases with Hostile Intent. *IEEE Software*, **20**, 58-66. http://dx.doi.org/10.1109/MS.2003.1159030

[8] Alexander, I. (2002) Modelling the Interplay of Conflicting Goals with Use and Misuse Cases. *Proceedings of* 8*th International Workshop on Requirements Engineering*: *Foundation for Software Quality* (*REFSQ*'02), Essen, 9-10 September 2002, 145-152.

[9] Korson, T. (1998) The Misuse of Use Cases (Managing Requirements).

[10] Pauli, J. and Xu, D. (2005) Misuse Case-Based Design and Analysis of Secure Software Architecture. *Proceedings of the International Conference on Information Technology Coding and Computing* (*ITCC*'05), Las Vegas, 4-6 April 2005, 398-403.

[11] Tohidi, M. (2003) Task Modeling. Directed Studies Research Honors Project. Carleton University, Ottawa.

[12] Smith, J. (1999) The Estimation of Effort Based on Use Cases. Rational Software White Paper.

[13] Sindre, G. and Opdahl, A. (2001) Capturing Security Requirements through Misuse Cases. *Proceedings of the* 14*th Norsk Information Conference* (*NIK*2001), Tromso, 26-28 November 2001, 212-221.

[14] Srivatanakul, T., Clark, J. and Polack, F. (2004) Writing Effective Security Abuse Cases. Technical Report YCS-2004-375. University of York, York.

[15] Constantine, L. and Lockwood, L. (2001) Structure and Style in Use Cases for User Interface Design. Object-Modeling and User Interface Design. Addison-Wesley, Boston.

[16] Neumann, P. (2004) Principle Assuredly Trustworthy Composable Architectures. CDRL A001 Final Report—DARPA.

[17] Allenby, K. and Kelly, T. (2001) Deriving Requirements Using Scenarios. *Proceedings of the* 5*th IEEE International Symposium on Requirements Engineering* (*RE*'01), Toronto, 27-31 August 2001, 228-235.

[18] Brown, D. and Densmore, J. (2005) The New, Improved RUP SE Architecture Framework. *IBM Rational*, **1**, 1-36.

[19] Bittner, K. and Spence, I. (2003) Use Case Modeling. Addison-Wesley, Boston.

[20] Pauli, J. and Xu, D. (2005) Threat-Driven Architectural Design of Secure Information Systems. *Proceedings of the* 7*th International Conference on Enterprise Information Systems* (*ICEIS*'05), Miami, 24-28 May 2005, 136-143.

[21] Pauli, J. and Xu, D. (2006) Threat-Driven Design and Analysis of Secure Software Architectures. *Journal of Information Assurance* (*JIAS*), **1**, 171-180.

[22] Pauli, J. and Xu, D. (2006) Ensuring Consistent Use/Misuse Case Refinement for Secure Systems. *Proceedings of the* 18*th International Conference on Software Engineering and Knowledge Engineering* (*SEKE* 2006), San Francisco, 5-7 July 2006, 392-397.

[23] Pauli, J. and Xu, D. (2006) Integrating Functional and Security Requirements with Use Case Refinement. *Proceedings of the* 11*th International Conference on Engineering of Complex Computer Systems* (*ICECCS* 2006), Stanford.

[24] Pfleeger, S. (2001) Software Engineering: Theory and Practice. 2nd Edition, Pearson Education, London.

[25] Eriksson, H., Penker, M., Lyons, B. and Fado, D. (2004) UML 2 Toolkit. Wiley, Indianapolis.

Scientific Research Publishing (SCIRP) is one of the largest Open Access journal publishers. It is currently publishing more than 200 open access, online, peer-reviewed journals covering a wide range of academic disciplines. SCIRP serves the worldwide academic communities and contributes to the progress and application of science with its publication.

Other selected journals from SCIRP are listed as below. Submit your manuscript to us via either submit@scirp.org or Online Submission Portal.