A Modeling Ontology for Integrating Vulnerabilities into Security Requirements Conceptual Foundations[⋆]

Golnaz Elahi[1], Eric Yu[2], and Nicola Zannone[3]

[1] Department of Computer Science, University of Toronto
gelahi@cs.toronto.edu
[2] Faculty of Information, University of Toronto
yu@ischool.utoronto.ca
[3] Eindhoven University of Technology
n.zannone@tue.nl

Abstract. Vulnerabilities are weaknesses in the requirements, design, and implementation, which attackers exploit to compromise the system. This paper proposes a vulnerability-centric modeling ontology, which aims to integrate empirical knowledge of vulnerabilities into the system development process. In particular, we identify the basic concepts for modeling and analyzing vulnerabilities and their effects on the system. These concepts drive the definition of criteria that make it possible to compare and evaluate security frameworks based on vulnerabilities. We show how the proposed modeling ontology can be adopted in various conceptual modeling frameworks through examples.

1 Introduction

Security needs are responses to being or feeling vulnerable. Vulnerable actors take measures to mitigate perceived risks, by using locks on the doors, surveillance cameras, etc. Existing security requirements engineering frameworks focus on various aspects for eliciting security requirements, such as attacker behavior [29, 31] and attacker goals [32], design of secure components [15], social aspects [18, 11], and events that can cause system failure [1]. However, attacks and consequent security failures often take place because of the exploitation of weaknesses or backdoors within the system. These weaknesses of the system or its environment that in conjunction with an internal or external threat can lead to a security failure are known as *vulnerabilities* [28] in security engineering. Vulnerabilities such as buffer overflow or weak passwords may result from misspecifications in the requirements, neglecting required pre- and post-conditions checks, faulty design and architecture, and programming errors.

In recent years, software companies and government agencies have become particularly aware of the security risks that vulnerabilities impose on system security and have started analyzing and reporting detected vulnerabilities of products and services [5, 6, 23, 27]. This empirical knowledge of vulnerabilities is used for scanning and maintaining system security and updating patches. However, vulnerability analysis has not

[⋆] Financial support from Natural Science and Engineering Research Council of Canada and Bell University Labs is gratefully acknowledged.

played a significant role in the elicitation of security requirements. There is evidence that knowing how systems have failed can help analysts build systems resistant to failures [24]. For this purpose, analysts should answer three basic questions [17]: (1) how a vulnerability enters into the system; (2) when it enters into the system; (3) where it is manifested in the system.

Vulnerabilities are introduced into the system by performing some activities or employing some assets. By identifying vulnerabilities and explicitly linking them to the activities and assets that introduce them into the system, analysts can recognize the vulnerable components of the system, study how vulnerabilities spread within the system, trace security failures back to the source vulnerability, and relate vulnerabilities to the stakeholders that ultimately are hurt. This information helps analysts understand how threats compromise the system, assess the risks of vulnerabilities, and decide on countermeasures to protect the system [9]. Some contributions [2, 17] collect and organize vulnerabilities and security flaws for providing analysts with more precise security knowledge. However, they do not provide a conceptual framework that allows analysts to elicit security requirements according to the identified vulnerabilities. To define a systematic way for linking empirical security knowledge, we need to identify the basic concepts that come into play when facing security issues. Those concepts influences the security analysis that analysts can perform.

This paper proposes a modeling ontology for integrating vulnerabilities into the security requirements conceptual foundations. We refer to the structure of conceptual modeling elements and their relationships as the conceptual foundation of a modeling framework. The proposed ontology, which is independent of the existing conceptual modeling foundations, aims to detect the missing security constructs in security requirements modeling frameworks and facilitates their enhancement. The ontology can be used as a unified way for comparing different conceptual foundations and their reasoning power as well as extending their ability for modeling and analyzing vulnerabilities. We propose the modeling ontology by means of a general meta-model. The meta-model helps integrate vulnerabilities into the conceptual foundation of a target framework, and the extended framework can be used for modeling and analyzing security requirements. To make the discussion more concrete, the proposed meta-model is adopted in three target conceptual frameworks, and the benefits and limitations of such adoptions are discussed.

The paper is organized as follows. Section 2 discusses the conceptual foundation for security analysis with a particular focus on vulnerabilities. Section 3 discusses and compares existing security frameworks centered on vulnerabilities. Section 4 introduces a vulnerability modeling ontology. Section 5 discusses how the modeling ontology can be realized in different target frameworks. Section 6 gives examples of integrating the ontology into three security requirements engineering frameworks. Finally, Section 7 draws conclusions and discusses future work.

2 The Conceptual Foundation for Vulnerability Analysis

This section reviews the security literature with the aim of defining a conceptual foundation for security requirements engineering centered on vulnerabilities. We discuss the basic security conceptual constructs together with the analysis facilities they offer.

A basic concept that comes into play when eliciting security requirements is the concept of *asset*. In security engineering, an asset is *"anything that has value to the organization"* [13]. Assets can be people, information, software, and hardware [7].

Assets and services can be the target of *attackers* (or *malicious actor*), and consequently, need to be protected. Attackers can be internal or external entities of the system. They perform *malicious actions* which attempt to break the security of a system or a component of a system. An *attack* is a set of intentional unwarranted (malicious) actions designed to compromise confidentiality, integrity, availability or any other desired feature of an IT system [30]. By analyzing the possible ways in which a system can be attacked, analysts can study attackers' behavior, estimate the cost of attacks, and determine their impact on system security.

Malicious actors often exploit *vulnerabilities* within the system to attack it. A vulnerability is a weakness or a backdoor which allows an attacker to compromise its correct behavior [28]. In the physical world, vulnerabilities are usually tangible and measurable. A crack in the wall is a concrete example of physical weakness. In the context of computer security, vulnerabilities are less tangible and visualizable. Vulnerabilities are *brought* to the system by adopting a software product or executing a service. By identifying the source of the vulnerability (e.g., software product, service, or data), analysts can identify what are the vulnerable components of the system, propagate the vulnerabilities in the model of the system, evaluate the benefits and risks of (vulnerable) entities, and decide on cost-effective countermeasures accordingly.

Risk has been proposed as a measure to evaluate the impact of an attack on the system. Risk involves the probability (*likelihood*) of a successful attack and its *severity* on the system [12]. Risk assessment is a type of analysis one can perform using security conceptual models. Therefore, risk is not a primitive concept and we do not include it into the meta-model for security requirements frameworks (Section 4).

Analyzing attacks and vulnerabilities allows analysts to understand how attackers can compromise the system. However, to assess the risk of an attack, analysts also need to consider the motivations (*malicious goals*) of attackers. Understanding why the attackers may attack the system helps identify the target of the attack and estimate the efforts (e.g., time, cost, resources, etc.) that attackers are willing to spend to compromise the system. Schneier [29] argues that understanding who are the attackers along with their motivations, goals, and targets, aids designers in adopting proper countermeasures to mitigate threats.

When the risk of an attack is higher than the risk tolerance of some stakeholder, analysts need to take the adequate measure to mitigate such risks [1]. A *countermeasure* is a protection mechanism employed to secure the system [30]. Countermeasures can be actions, processes, devices, solutions, or systems, such as firewalls, authentication protocols, digital signature, etc. Knowledge about attackers' behavior and vulnerabilities helps analysts in the identification of appropriate countermeasures to protect the system. Countermeasures intend to prevent attacks or vulnerability exploitations from compromising the system. For instance, they are used to patch vulnerabilities or prevent their exploitation. Modeling and analyzing the countermeasures is important for evaluating their efficacy and consequently the ultimate security of the system.

Several conceptual modeling frameworks for security analysis take advantage of *temporally-ordered* models for analyzing attacks [21, 25]. Incorporating the concept of time into the attack modeling helps understand the sequence of actions and vulnerability exploitations which lead to a successful attack. The resulting model is useful for analyzing attacks as well as designing and evaluating countermeasures that prevents the attacks at the right step. On the other hand, temporally-ordered models of the system and stakeholders' interactions increase the complexity of requirements models which may not be suitable for the early stages of the development.

3 Vulnerability Modeling and Analysis Approaches

This section surveys and compares different approaches proposed in the literature for modeling, organizing, and analyzing vulnerabilities. We also discuss the types of reasoning that the existing conceptual frameworks support.

3.1 Vulnerability Catalogs

The most primitive way for modeling and organizing vulnerabilities is grouping detected and reported flaws and weaknesses into catalogs. Although catalogs are not conceptual models, they are not entirely structure-less. Various web-based software vulnerability knowledge bases provide searchable lists of vulnerabilities. Catalogs of vulnerabilities contain different types of information with different information granularity which are useful for specific stages of the development and types of analysis. These web portals aim to increase the level of awareness about vulnerable products and severity of vulnerabilities. For example, the National Vulnerability Database [27], SANS top-20 annual security risks [27], and Common Weakness Enumeration (CWE) [6] provide updated lists of vulnerabilities and weaknesses. CVE contains vendor-, platform- and product-specific vulnerabilities. SANS list and CWE catalog include more abstract weaknesses, errors, and vulnerabilities. Some entries in these lists are technology and platform independent, while some of the vulnerabilities are described for specific product, platform, and programming language.

3.2 Vulnerability Analysis for Computer Network Security

Modeling and analyzing vulnerabilities within computer networks is common, because vulnerabilities in such systems can be easily associated to physical nodes of the network. Several attack modeling and analysis approaches [25, 19, 10] take advantage of Attack Graphs and Bayesian Networks for vulnerabilities assessment at the network level. Phillips et al. [25] introduce Attack Graphs to analyze vulnerabilities in computer networks. Attack graphs provide a method for modeling attacks and relating them to the machines in a network and to attackers. Liu and Man [19] use Bayesian Networks to model all potential atomic attack steps in a network. Causal relationships between vulnerabilities encoded in an attack graph are used to model the overall security of a network in [10].

3.3 Modeling Vulnerabilities for Security Requirements Engineering

In secure software engineering frameworks, vulnerabilities usually refer to the general openness to attacks and risks. For example, Liu et al. [18] propose a vulnerability analysis method for eliciting security requirements, where vulnerabilities are the weak dependencies that may jeopardize the goals of depender actors in the network of social and organizational dependencies.

Only a few software engineering approaches consider analyzing vulnerabilities, as weaknesses of the system, during the elicitation of security requirements. Matulevicius et al. [20] treat vulnerabilities as beliefs in the knowledge base of attackers which may contribute to the success of an attack. In [22], the i* framework is extended to represent vulnerabilities and their relation with threats and other elements of the i* models.

The CORAS project [7] proposes a modeling framework for model-based risk assessment in the form of a UML profile. The profile defines UML stereotypes and rules to express assets, risks that target the assets, vulnerabilities, accidental and deliberate threats, and the security solutions. CORAS provides a way for expressing how a vulnerability leads to another vulnerability and how a vulnerability or combination of vulnerabilities lead to a threat. CORAS also provides the means to relate treatments to threats and vulnerabilities.

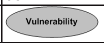
Rostad [26] suggests extending the misuse case notation for including vulnerabilities into requirements models. Vulnerabilities are defined as a weakness that may be exploited by misuse cases. Vulnerabilities are expressed as a type of use case, with an *exploit* relationship from the misuse case to the vulnerability and an *include* relation with the use case that introduces the vulnerability.

3.4 Comparison of the Conceptual Modeling Frameworks

Table 1 compares capabilities of the reviewed conceptual structures based on the conceptual foundation discussed in Section 2. The conceptual modeling frameworks that focus on security requirements engineering, model vulnerabilities in various ways. Among them, CORAS [7] does not investigate which design choices, requirements, or processes have brought the vulnerabilities to the system, and the semantics of relationships among vulnerabilities, and between vulnerabilities and threats are not defined. Similar to CORAS, the resulting models in [20, 22] do not specify how, by what actions and actors the vulnerability is brought to the system. These models do not capture the impact of countermeasures on the vulnerabilities and attacks. In [22], threats are not related to the attacker that poses them, and the semantics of the relation between threats and vulnerabilities is not well defined.

In summary, the missing point in the surveyed approaches is lack of modeling constructs that express how vulnerabilities enter into the system and how they spread out within the system. The link between attacks and vulnerabilities are implicitly (and explicitly) modeled in all of the surveyed approaches. However, among the modeling notations that provide explicit constructs for modeling vulnerability, only a few frameworks

Table 1. Comparison of modeling notations. N indicates that the concept or relation is not considered, and Y indicates the relation is considered explicitly in the notation. P means the relation is implicitly considered or its semantics is not well defined.

Method	Web-based vulnerabilities knowledge sources	Network security analysis methods	CORAS Framework [3]	Secure Tropos by Matulevicius et al. [20]	Risk-Based Security Framework by Mayer et al. [22]	Extensions to misuse case diagram [26]	Security extension on i* framework by Elahi et al. [8, 9]
Conceptual Foundation	Structured and searchable catalogs	Network configuration models, AG, BN	CORAS UML-profile based models	Secure Tropos	i* framework	Misuse case models	i* framework
Vulnerability graphical representation	None	None					●
Relation of vulnerabilities to vulnerable elements	N	Y	N	N	N	P	Y
Relation of vulnerabilities to other vulnerabilities	Y	Y	P	N	N	N	N
Propagation of vulnerabilities to other system elements	N	Y	N	N	N	N	Y
Effects of vulnerabilities	Y	Y	Y	Y	P	N	Y
Severity of vulnerabilities	Y	Y	N	N	N	N	Y
Relation of vulnerabilities and attacks (exploitation)	P	Y	P	P	Y	Y	Y
Countermeasures' impacts on vulnerabilities	N	P	P	N	N	Y	Y
Steps of vulnerability exploitation (sequence)	N	Y	N	N	N	N	N

such as CORAS [7], i* security extensions [9, 8], and extensions of misuse case models [26] relate the countermeasures to vulnerabilities. The semantics of the countermeasure impact in [7, 26] is not well defined, and the model cannot be used to evaluate the impact of countermeasures on the overall system security. Although modeling and analyzing the order of actions to accomplish an attack may affect the countermeasure selection and development, the existing frameworks for security requirements engineering do not consider the concept of sequence (temporal order) in their meta-model.

4 A Modeling Ontology for Vulnerabilities

This section presents a vulnerabilities modeling ontology which aims to incorporate vulnerabilities into requirements models for expressing how vulnerabilities are brought to the system and propagated, how the vulnerabilities get exploited by attackers and affect different actors, and how countermeasures mitigate the vulnerabilities. The ontology is described by an abstract meta-model, which defines and relates the conceptual constructs gathered in Section 2.

Fig. 1 depicts the proposed vulnerability-centric meta-model. The conceptual modeling framework that one may integrate with ontology elements is called the *target* framework. The target framework can be business process modeling frameworks, UML static and dynamic diagrams, agent- and goal-oriented modeling frameworks, etc.

Vulnerability Definition in the Ontology. A *concrete element* is a tangible entity. Depending on the target framework, the concrete element can be an activity, task, function, class, use case, etc. Concrete elements may introduce vulnerabilities into the system, which are then called *vulnerable elements*. In the meta-model the link between a vulnerability and a concrete element is captured by the *bring* relation. Exploitation of vulnerabilities can have *effects* on other elements These elements are called *affected*

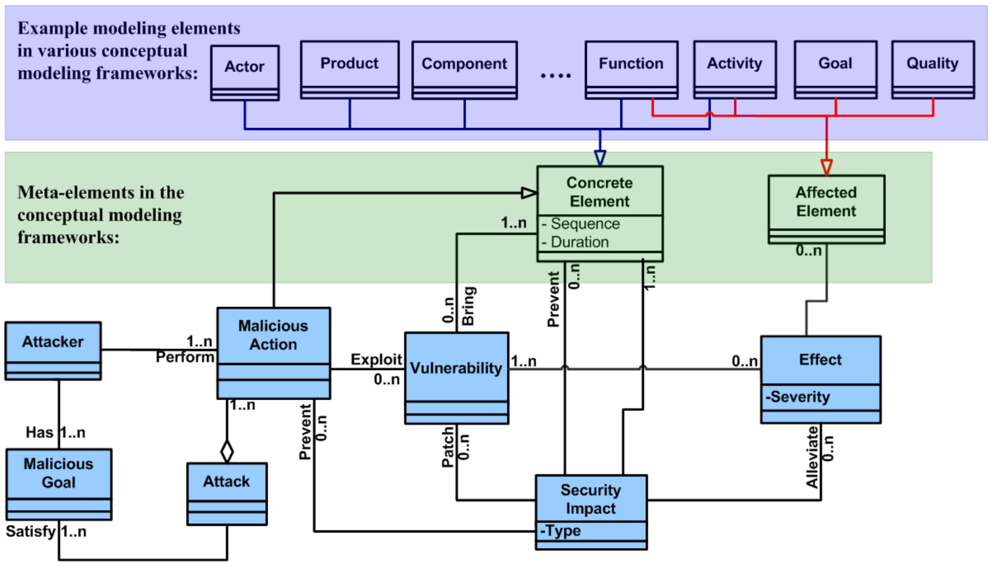


Fig. 1. The vulnerability-centric modeling ontology for security concepts

elements. The *effect* relation is presented as a class and is characterized by the attribute *severity* that specifies the criticality of vulnerabilities effects.

Attack and Attacker Definition in the Ontology. An *attack* involves the execution of (a *sequence* of) *malicious actions* that one or more actors perform to satisfy some *malicious goal*. Linking attackers to malicious actions allows modeling attacks that require the collaboration of different attackers. A *malicious action* can *exploit* a number of vulnerabilities, which has (negative) effects on the *affected element*s. This negative effect is captured as a relation which links vulnerabilities to the affected elements. This relation is modeled as a class in the meta-model, which enables defining the severity of the effect as an attribute of the class.

Countermeasure Definition in the Ontology. A concrete element may have a *security impact* on attacks. Such an element can be interpreted as a security countermeasure. The *security impact* is a relationships which is expressed as a class in the meta-model. Security countermeasures can be used to *patch* vulnerabilities, *alleviate* the effect of vulnerabilities, *prevent* the malicious actions that exploit vulnerabilities or can prevent (or remove) the concrete elements that bring the vulnerabilities. By patching a vulnerability, the countermeasure fixes the weakness in the system. Examples of such a countermeasure is a software update that the vendors provide. A countermeasure that alleviates vulnerability effects, does not address the source of the problem, but it intends to reduce the effects of the vulnerability exploitation. For example, a backup system alleviates the impact of security failures that cause data loss. Countermeasures can also prevent an attacker to perform some actions. For example, an authentication solution

Table 2. The mapping of the elements in the vulnerability modeling ontology to elements of different modeling elements. The x in the cells indicate that the target framework does not provide any embedded element for the element of ontology and a new modeling construct is required.

	Static models (UML Class diagram)	Dynamic models (UML Sequence diagram)	Requirements models (UML use case diagram)	Goal modes (i* agent- and goal-oriented model)
Vulnerability	x (New Element)	x (New Element)	x (New Element)	x (New Element)
Concrete Element	Classes, Packages, Operations, Attributes	Messages, Guards, Combined Fragments	Use cases	Tasks, Resources
Attacker	x	Roles	Actors (misuser)	Actors
Malicious Action	x	Concrete elements for modeling behavior	Misuse Cases	Tasks
Malicious Goals	x	x	x	Goals
Effect	x	x	Adding new Stereotypes	Contribution Links
Affected Element	Classes, Packages, Operations, Attributes	Messages, Guards, Combined Fragments	Use Cases	Goals, Tasks, Resources
Security Impact	x	x	Adding new Stereotypes	Using and extending Contribution Links

prevents unauthorized access to assets. Countermeasure may prevent performing vulnerable actions or using vulnerable assets, which results in removing the vulnerable elements that have brought vulnerabilities to the system. For example, disabling JavaScript option in the browser prevents the browser to run a malware.

5 The Adoption of the Modeling Ontology

In the previous section, we defined the modeling ontology that can be used to integrate vulnerabilities into existing conceptual modeling frameworks. This section discusses the adoption and realization of the proposed modeling ontology in various types of conceptual modeling frameworks. Table 2 provides a mapping between the modeling constructs in four example conceptual modeling frameworks and the elements of the vulnerability-centric modeling ontology. The mapping illustrates which modeling constructs in the frameworks can be used (or inverted) for expressing the ontology's elements, and which elements of the ontology need to be incorporated in the target conceptual framework by adding a new construct. In this table, UML class and sequences diagrams are examples of static and dynamic modeling approaches, respectively. Use case and i* models are examples of requirements models. The comparison can be generalized to other similar conceptual frameworks (e.g., the properties for sequence diagrams can be generalized to other dynamic modeling approaches).

Realization of Vulnerabilities in the Target Framework. To incorporate *vulnerabilities* into a target framework, a new modeling construct (with a graphical representation) need to be added to the target framework. Vulnerabilities need to be (graphically) linked to the vulnerable element, which expresses the *bring* relationship. The vulnerability effect and its severity need to be defined in each specific conceptual modeling framework according to the semantics of relationships in that conceptual framework. For example, in the UML use case diagram, one may define a new stereotype to specify the effect of vulnerabilities exploitation (and its severity), and in a goal-oriented modeling framework like i*, contribution links can be used to represent the effect of vulnerabilities and their severity. Existing relationship in static and dynamic modeling approaches do not provide the required semantics to model the vulnerability effects.

Modeling vulnerabilities (and related concepts) in different conceptual modeling frameworks facilitate different types of analysis and reasoning. Adding vulnerabilities to static models such as deployment diagrams allows one to propagate vulnerabilities from the elements that bring the vulnerabilities to other system components, by analyzing the function that vulnerable components play in the system. By integrating vulnerabilities into dynamic models, one can detect the sequence of vulnerability propagation in a period of time. Integration of vulnerabilities into requirements and goal models help detect the functionalities that introduce risks to the system (by bringing vulnerabilities). In addition, vulnerabilities can be propagated into the network of functions, goals, and actors. Examples of vulnerabilities propagation can be found in [9].

Realization of Attacks and Attackers in the Target Framework. The definition of attacks is fundamentally a matter of perspective: the nature and semantics of *malicious actions* are similar to the nature of conceptual elements that model the normal behavior of the system. Therefore, distinguishing the malicious and non-malicious behavior does not affect the analysis one can perform on the models. However, Sindre and Opdahl [16] show that graphical models become much clearer if the distinction between malicious and non-malicious elements is made explicit and the malicious actions are visually distinguished from the legitimate ones. They show that the use of inverted elements strongly draws the attention to dependability aspects early on for those who discuss the models. Therefore, to model *malicious actions* in the target frameworks, the (inverted) concrete elements that model normal actions and interactions within the system is semantically sufficient. For example, in a sequence diagram, the sequences of messages to mount an attack can be modeled using the existing sequence modeling constructs.

Several conceptual modeling frameworks provide the required foundations for modeling sequence of actions in a temporally-ordered fashion (e.g., sequence diagrams, state charts, activity diagrams). On the other hand, the modeling approaches that provide a static view to the system, such as UML class, deployment, package or component diagrams, and data models, do not support modeling actions and dynamic behavior of the system. Such frameworks are not expressive enough for modeling malicious actions. Some conceptual frameworks provide means to model the system and actors' actions in a static way (e.g., use case diagrams and i* agent- and goal-oriented models). Such modeling approaches provide a static view of the malicious actions and vulnerability exploitations, and cannot model the temporally-ordered sequence of actions or messages, vulnerability exploitations, and pre-conditions that lead to an attack.

Attackers can be modeled using the (inverted) actor element in the target framework. For example, an attacker can be a role with a lifeline in UML sequence diagrams or an actor that triggers misuse cases in use case diagrams. However, some conceptual modeling frameworks, such as UML class or deployment diagrams do not provide constructs for expressing actors, which limits the security analysis that they can perform.

Several conceptual modeling frameworks focus on "what" and "how" in the system. Such frameworks, such as UML static and dynamic diagrams, do not allow modeling the intentions and motivations of the interacting parties in the system. Goal-oriented conceptual modeling frameworks such as i*, Tropos, and KAOS provide required means to model goals; therefore, the attackers' malicious goals can be modeled by using (inverted) conceptual constructs that these frameworks provide for modeling goals of interacting parties.

Realization of Countermeasures in the Target Framework. We do not distinguish security elements from non-security elements in the meta-model, because the nature of elements which specify the system behavior is not different from the elements that model the security mechanisms of the system, and the distinction does not affect the security requirements analysis. Similar to the vulnerabilities' effects, the semantics of countermeasures' impact need to be defined in each specific conceptual modeling framework according to the semantics of relationships in the target framework.

6 Examples of Adopting the Proposed Ontology

In this section, the proposed ontology is adopted in three conceptual foundations to illustrate the realization of the ontology and its benefits. These examples aim to illustrate how the elements of the meta-model are realized in different conceptual frameworks for (security) requirements and risk analysis. We integrate the concept of vulnerability into misuse case models, as an example of a static requirements modeling approach. We revise CORAS, as an example of risk analysis frameworks which is able to express vulnerabilities. In this example, we analyze how the adoption of the ontology can enhance its reasoning and analysis power. Finally, we show how vulnerabilities and related security concepts can be added to the i* framework, as an example of goal-oriented requirements modeling frameworks. All the enhancements are illustrated with the meta-model and concrete examples based on a browser and web application scenario.

6.1 Integrating Vulnerability Modeling in (Mis)Use Case Diagrams

Misuse case analysis is known as a useful technique for eliciting and modeling security requirements and threats [31]. In misuse case models, attacks and attackers are expressed using inverted use cases and actors, where misuse cases threaten other use cases and security use cases mitigate the attacks. However, misuse case models do not capture the vulnerabilities that attackers may exploit to compromise the system. In addition, models are not expressive enough to fully capture the impact of security uses case on other (mis)use cases. For instance, one can only model countermeasures that prevent misuse cases, whereas countermeasures for patching vulnerabilities and alleviating their exploitation impact cannot be represented.

Fig. 2 shows the revised meta-model of misuse case models by adopting the proposed modeling ontology to fill the discussed gaps. In the meta-model, the element and relationships added from the ontology are represented as highlighted classes and dashed relationships, respectively. The concrete elements in the use case models is the "use case" element which may *bring* vulnerabilities to the system. An attack (misuse case) exploits a vulnerability, and the effect of the exploitation is a *threaten* relation to

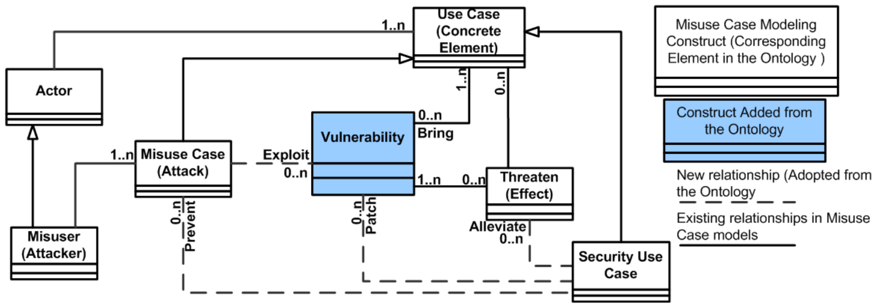

Use Case (Concrete Element)
Actor
Misuse Case (Attack)
Misuser (Attacker)
Vulnerability
Threaten (Effect)
Security Use Case
1..n
Exploit
0..n
Bring
Prevent
Patch
Alleviate
Misuse Case Modeling Construct (Corresponding Element in the Ontology)
Construct Added from the Ontology
New relationship (Adopted from the Ontology
Existing relationships in Misuse Case models

Fig. 2. Revising the misuse case modeling notation by adopting the modeling ontology

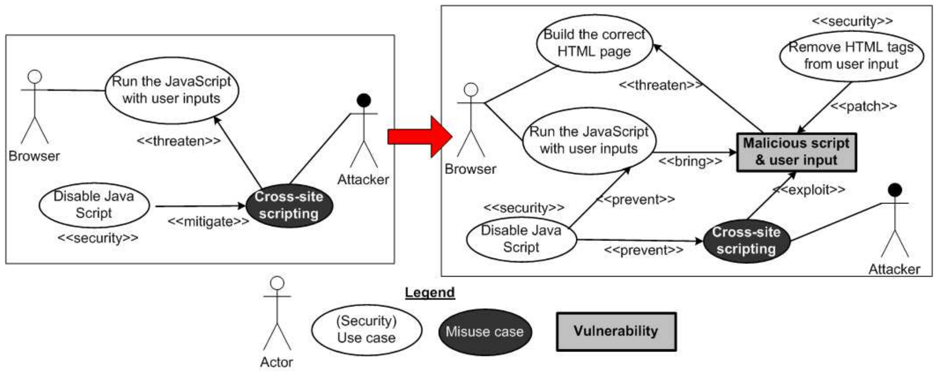


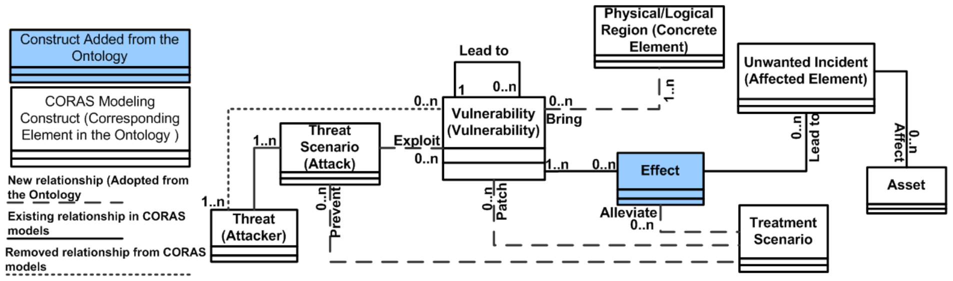
Fig. 3. Integrating vulnerabilities into the misuse case diagrams, example of a web application and brower scenario

other use cases. New relationships such as *exploits* and *effects* of security use cases are modeled by new stereotypes. Fig. 3 depicts the adoption of some of the ontology elements into the misuse case diagrams. The left hand side of the figure shows misuse case models [31] for a web application scenario where a cross site scripting attack occurs, and the right hand side of the model shows our proposal for modeling vulnerabilities and linking them to (mis)use cases.

6.2 Revising Vulnerability Modeling in the CORAS Approach

CORAS [7] provides modeling constructs to express threats, vulnerabilities, threat scenarios, unwanted incidents, risks, assets, and treatment scenarios. CORAS models show the causal relationships from the vulnerabilities to threat scenarios; however, CORAS models do not show what actions or scenario in the system introduce vulnerabilities. The exploit relationship is not explicitly expressed, and the models do not express the effects of vulnerabilities' exploitation explicitly. Besides, treatment scenarios are only connected to vulnerabilities and the semantics of this relationship is not well defined.

Fig. 4 shows the revised meta-model of CORAS by adoption of the proposed vulnerability modeling ontology. In this meta-model, the elements and relationships that are adopted from the ontology are represented as highlighted classes and dashed rela-



Construct Added from the Ontology
CORAS Modeling Construct (Corresponding Element in the Ontology)
New relationship (Adopted from the Ontology
Existing relationship in CORAS models
Removed relationship from CORAS models
Threat (Attacker)
Threat Scenario (Attack)
Vulnerability (Vulnerability)
Physical/Logical Region (Concrete Element)
Unwanted Incident (Affected Element)
Effect
Asset
Treatment Scenario
Lead to
Exploit
Bring
Prevent
Patch
Alleviate
Affect
1..n
0..n
1

Fig. 4. Revising the CORAS risk modeling language by adopting the modeling ontology

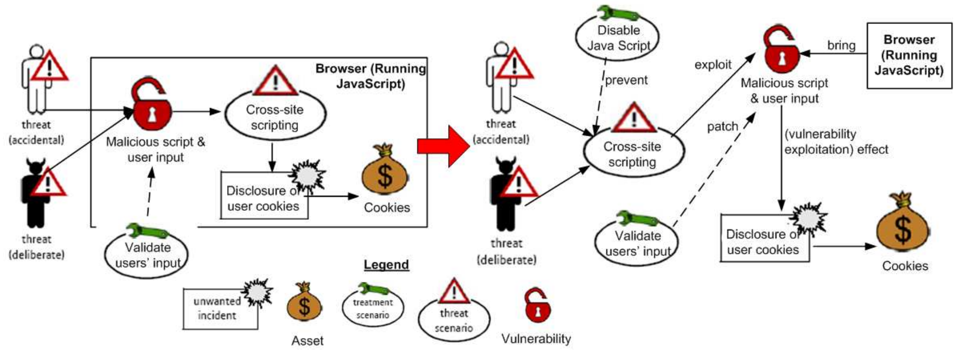


Fig. 5. Revising vulnerability modeling in the CORAS risk modeling approach, example of a web application and brower scenario.

tionships, respectively. The right hand side of Fig. 5 gives an example of adopting the proposed ontology in the graphical CORAS modeling language for the browser and web application case study, which is modeled using CORAS notation at the left hand side of the Fig. 5. The logical or physical region boxes are used as concrete elements; for example, the *browser* brings the vulnerability of *malicious script and user input*. Threatening actors and threat scenarios (*Cross-site scripting*) are directly connected, and the relationship between threat scenario and vulnerabilities is reversed. The exploitation effects and countermeasures impacts are modeled using the existing CORAS relationships with additional tags. Treatments (*validate users' input* and *disable JavaScript*) patch the vulnerabilities, prevent threat scenarios or alleviate the effect of vulnerabilities.

6.3 Integrating Vulnerabilities into the i* Framework

The ability of the i* framework [33] to model agents, goals, and their dependencies makes it suitable for understanding security issues that arise among multiple malicious or non-malicious social agents with competing goals. Thus, i* provides the basic elements for incorporating vulnerabilities into security requirements models and representing their propagation within the system.

Fig. 6 presents a fragment of the i* meta-model integrated with the vulnerability ontology and extended with malicious elements. The concrete elements in the i* frame-

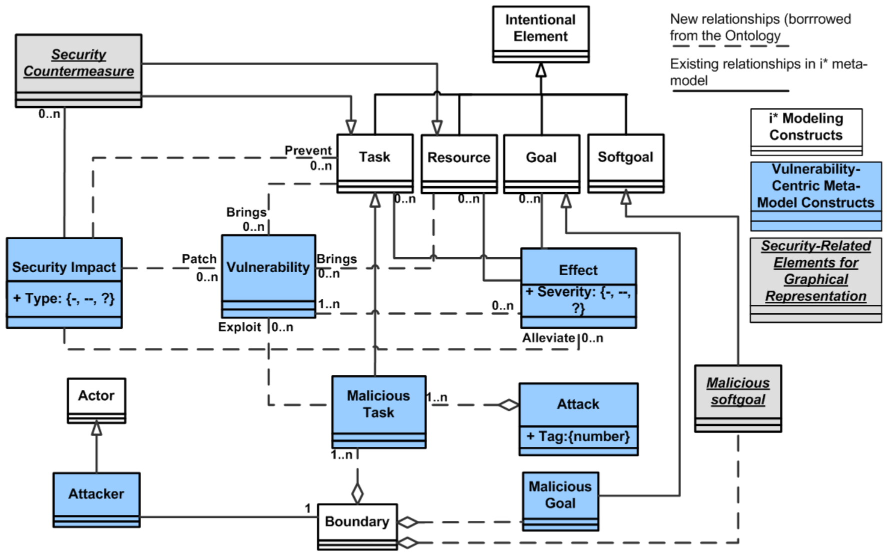


Fig. 6. The fragment of the i* meta-model extended by adopting the modeling ontology [9]

work that may bring vulnerabilities are tasks and resources. The effect of vulnerabilities and its severity in the i* framework are defined as Hurt (−), Break (−−), and Unknown (?) contribution links. Malicious tasks, goals, softgoals, and attackers are specialization of i* tasks, goals, softgoals, actors. Some tasks and resources can work as security countermeasures.

Fig. 7 shows how vulnerabilities and related security constructs are graphically integrated into i* models in the browser and web application example. To graphically represent vulnerabilities (*Malicious script*), the i* notation is enriched with a "black circle". The proposed notation graphically distinguishes malicious and non-malicious elements using a black shadow in the background of malicious elements as proposed in [18, 8]. The exploitation of a vulnerability by an attacker is represented by a link labeled *exploit* from the malicious task to the vulnerability. The exploitation of (a combination of) vulnerabilities has effects on goals, tasks, and availability of resources. Countermeasures are modeled using ordinary task elements, and their impacts as contribution links with *alleviate*, *prevent*, or *patch* tags. Detailed models and the goal model evaluation reasoning on the browser and web application case study can be found in [9].

6.4 Lessons Learned

The adoption of the proposed vulnerability modeling ontology in different conceptual foundations helps understanding the limitations of the conceptual foundations and facilitates their enhancement. The enhanced misuse case models provide additional information about vulnerabilities that enables a finer-grained security analysis for deciding on proper security use cases. The revised CORAS models explicitly express which threat scenario exploits the vulnerabilities and what are the effects of each exploitation, while

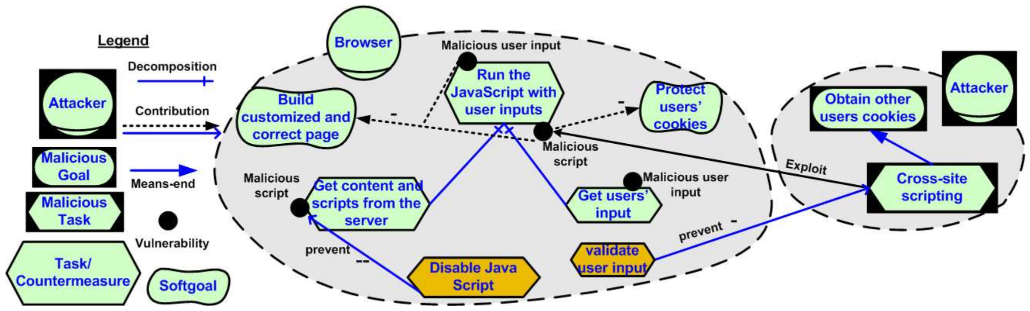


Fig. 7. Graphical representation of vulnerabilities in i* models.

the original CORAS models only express the impacts of the whole scenario. The additional tags for expressing the exploitation effects and countermeasures impacts make the semantics of CORAS relationships explicit. Analyzing the effects of vulnerabilities in the i* models allows one to assess the risks of attacks, analyze the efficacy of countermeasures, and decide on patching or disregarding the vulnerabilities by taking advantage of goal model evaluation techniques [4]. In particular, analysts can verify whether stakeholders' goals are satisfied with the risks of vulnerabilities and attacks, and assess the efficacy of security countermeasures against such risks. In addition, the resulting security goal models and goal model evaluation can provide a basis for trade-off analysis among security and other quality requirements [8].

However, conceptual foundations may not be suitable or expressive enough to model all the ontology elements. Each conceptual foundation has been proposed for a specific purpose and is suitable for a certain type of modeling and analysis. For instance, misuse cases and CORAS do not provide constructs to represent delegations of assets and dependencies between actors. Therefore, they cannot model and analyze the propagation of vulnerabilities to system components. In addition, misuse cases and CORAS models cannot express why a misuser attacks the system and link the misuser's actions to his/her goals. Another limitation of i*, misuse case, and CORAS models is lack of constructs to model temporally-ordered actions and vulnerabilities exploitations that lead to an attack. Enhancing these conceptual foundations to address above limitations require a deep restructuring of their conceptual foundation, which imposes a trade-off between complexity of models and their reasoning power. Therefore, analysts need to identify the objectives of their analysis and select the target framework accordingly. For instance, it may be more appropriate to extend a dynamic modeling approach such as sequence diagrams rather than adding temporal constructs to misuse case diagrams.

7 Conclusions and Future Work

This paper has proposed a modeling ontology for integrating vulnerabilities into conceptual modeling frameworks. In the process of the ontology development, we reviewed the security engineering and security requirements engineering literature to identify the set of core concepts needed for security requirements elicitation. The ontology is defined as an abstract meta-model which relates the elements of any conceptual framework to vulnerabilities and related security concepts. We also discussed how the ontology can be adopted and realized in different conceptual modeling frameworks through some examples. These examples show that different frameworks have different conceptual structure and capabilities; therefore, by adopting the ontology elements into each conceptual framework, different types of analysis can be done based on the resulting models. We found that since some conceptual modeling frameworks do not provide the required structures, they are not able to express concepts such as malicious goal, vulnerable element of the system, temporal order, etc.

We adopted the ontology in the misuse case diagrams, i* models, and CORAS risk models. In addition to those examples, in future work, the proposed ontology needs to be adopted into a wider variety of modeling frameworks to provide stronger empirical evidences for usefulness, expressiveness, and comprehensiveness of the ontology. In order to evaluate the proposed ontology, we are performing empirical studies including case studies with human subjects that use the extended conceptual modeling frameworks. The aim of such case studies is to discover the security related concepts or types of analysis that the elements of the ontology cannot express or human subjects have difficulties to express. We aim to interview the subjects and critically analyze the models to draw conclusions about the expressiveness of the proposed conceptual elements.

An issue not explored in this paper is the scalability concerns that come with graphical visualization of complex models. The resulting models extended with security concepts, may become complex and hard to understand. In order to manage the complexity, defining views of the system and filtering some views would be necessary.

References

1. Y. Asnar, R. Moretti, M. Sebastianis, and N. Zannone. Risk as Dependability Metrics for the Evaluation of Business Solutions: A Model-driven Approach. In *Proc. of DAWAM'08*, pages 1240–1248. IEEE Press, 2008.
2. A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *TDSC*, 1(1):11–33, 2004.
3. F. Braber, I. Hogganvik, M. S. Lund, K. Stolen, and F. Vraalsen. Model-based security analysis in seven steps — a guided tour to the coras method. *BT Technology Journal*, 25(1):101–117, 2007.
4. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, editors. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishing, 2000.
5. Common Vulnerability Scoring System. http://www.first.org/cvss/.
6. Common Weakness Enumeration. http://cwe.mitre.org/.
7. F. den Braber, T. Dimitrakos, B. A. Gran, M. S. Lund, K. Stolen, and J. O. Aagedal. The CORAS methodology: model-based risk assessment using UML and UP. In *UML and the unified process*, pages 332–357. IGI Publishing, 2003.
8. G. Elahi and E. Yu. A goal oriented approach for modeling and analyzing security trade-offs. In *Proc. of ER'07*, LNCS 4801, pages 375–390. Springer, 2007.
9. G. Elahi, E. Yu, and N. Zannone. A vulnerability-centric requirements engineering framework: Analyzing security attacks, countermeasures, and requirements based on vulnerabilities. *Manuscript submitted to Req. Eng. Journal*, 2009.
10. M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring network security using dynamic bayesian network. In *Proc of QoP'08*, pages 23–30. ACM, 2008.

11. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling security requirements through ownership, permission and delegation. In *Proc. of RE'05*, pages 167–176. IEEE Press, 2005.
12. ISO/IEC. Risk management-vocabulary-guidelines for use in standards. ISO/IEC Guide 73, 2002.
13. ISO/IEC. Management of Information and Communication Technology Security – Part 1: Concepts and Models for Information and Communication Technology Security Management. ISO/IEC 13335, 2004.
14. S. Jajodia. Topological analysis of network attack vulnerability. In *Proc. of ASIACCS'07*, pages 2–2. ACM, 2007.
15. J. Jürjens. *Secure Systems Development with UML*. Springer, 2004.
16. J. Krogstie, A. L. Opdahl, and S. Brinkkemper. Capturing dependability threats in conceptual modelling. *Conceptual Modelling in Information Systems Engineering*, pages 247–260, 2007.
17. C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi. A taxonomy of computer program security flaws. *CSUR*, 26(3):211–254, 1994.
18. L. Liu, E. Yu, and J. Mylopoulos. Security and privacy requirements analysis within a social setting. In *Proc. of RE'03*, page 151. IEEE Press, 2003.
19. Y. Liu and H. Man. Network vulnerability assessment using bayesian networks. In *Data mining, intrusion detection, information assurance, and data networks security*, pages 61–71. Society of Photo-Optical Instrumentation Engineers, 2005.
20. R. Matulevicius, N. Mayer, H. Mouratidis, E. Dubois, P. Heymans, and N. Genon. Adapting Secure Tropos for Security Risk Management in the Early Phases of Information Systems Development. In *Proc. of CAiSE'08*, pages 541–555, 2008.
21. J. P. McDermott. Attack net penetration testing. In *Proc. of NSPW'00*, pages 15–21. ACM, 2000.
22. N. Meyer, A. Rifaut, and E. Dubois. Towards a Risk-Based Security Requirements Engineering Framework. *In Proc. of REFSQ'05*, 2005.
23. National Vulnerability Database. http://nvd.nist.gov/.
24. H. Petroski. *To Engineer is Human: The Role of Failure in Successful Design*. St. Martin's Press, New York, 1985.
25. C. Phillips and L. P. Swiler. A graph-based system for network-vulnerability analysis. In *Proc. of NSPW'98*, pages 71–79. ACM, 1998.
26. L. Rostad. An extended misuse case notation: Including vulnerabilities and the insider threat. In *Proc. of REFSQ'06*, 2006.
27. SANS. http://www.sans.org/.
28. F. B. Schneider, editor. *Trust in Cyberspace*. National Academy Press, 1998.
29. B. Schneier. Attack trees. *Dr. Dobb's Journal*, 24(12):21–29, 1999.
30. B. Schneier. *Beyond Fear*. Springer, 2003.
31. G. Sindre and L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.
32. A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proc. of ICSE'04*, pages 148–157. IEEE Press, 2004.
33. E. Yu. *Modeling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, 1995.