ABSTRACT

Misuse cases are a way of modeling negative requirements, that is, behaviors that should not occur in a system. In particular, they can be used to model attacks on a system as well as the security mechanisms needed to avoid them. However, like use cases, misuse cases describe requirements in a high-level and informal manner. This means that, whilst they are easy to understand, they do not lend themselves to testing or analysis. In this paper, we present an executable misuse case modeling language which allows modelers to specify misuse case scenarios in a formal yet intuitive way and to execute the misuse case model in tandem with a corresponding use case model. Misuse scenarios are given in executable form and mitigations are captured using aspect-oriented modeling. The technique is useful for brainstorming potential attacks and their mitigations. Furthermore, the use of aspects allows mitigations to be maintained separately from the core system model. The paper, supported by a UML-based modeling tool, describes an application to two case studies, providing evidence that the technique can support red-teaming of security requirements for realistic systems.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications – Elicitation methods, Languages

General Terms
Security, Languages

Keywords
Misuse cases, Scenarios, Early Aspects

1. INTRODUCTION

A misuse case [1, 2] is “a use case from the point of view of an actor hostile to the system under design” [1]. Misuse cases have been used effectively to model potential attacks to a system and to analyze the trade-offs of mitigation strategies [3]. However, misuse cases, just like use cases, are an informal notation and therefore are not directly amenable to formal analysis or testing. This paper presents an executable modeling technique for misuse cases that allows modelers to specify use cases and misuse cases for a system, and then to animate those misuse cases on the use case model. This gives project stakeholders a way to brainstorm potential attacks, model those attacks as misuse cases, model mitigation mechanisms to prevent the attacks, and then animate or test the resulting model to ensure that the attacks are mitigated appropriately. Furthermore, the technique can be used in eliciting new attacks and animating the effects of those attacks on the system behavior. The executable modeling language has a formal definition, yet is based on UML and so remains accessible to stakeholders.

The approach builds upon recent work in software modeling, namely, precise use case modeling [4, 5], aspect-oriented modeling [6-8], and statechart synthesis from scenarios [9]. Use and misuse case scenarios are modeled precisely using extended interaction overview diagrams (EIODs) [5]. Attack mitigation scenarios are modeled as aspects that weave mitigation behavior into the use case model. The entire weaved scenario model can then be transformed automatically into a set of finite state machines (FSMs). These FSMs can be animated or the misuse scenarios can be executed on the FSMs to check that the mitigations do indeed prevent the attacks.

EIODs are a formalization of UML interaction overview diagrams for modeling scenarios precisely. In this paper, we extend EIODs to the modeling of misuse cases by including use/misuse case relationships. Aspect-oriented modeling is a way of separating cross-cutting concerns during software modeling. In this paper, we model attack mitigations as aspect scenarios. This is beneficial because it maintains a clear separation, during requirements modeling, of the core functionality of the system and functionality needed to handle security concerns. The mitigating behavior can be automatically weaved with the use case behavior to obtain a scenario-based description of the complete system. Finally, we use the statechart synthesis algorithm from [9] to make the combined system behavior executable. Statechart synthesis automatically generates a set of communicating FSMs from an EIOD. Since FSMs have a well understood semantics, they can be executed, animated and/or analyzed.

The contribution of this paper, therefore, can be measured in two ways. The first contribution is the integration of these three
techniques applied to executable misuse case modeling. Secondly, the paper extends two of these three techniques. EIODs are extended to support misuse case modeling and aspect modeling is extended to support expressive weaving strategies for scenarios. The first extension is reasonably straightforward; the second is significant.

The ideas in the paper have been implemented in a tool, MUCSIM (misuse case simulator), which is implemented as a plug-in to IBM Rational Software Modeler. MUCSIM integrates two existing tools, UCSIM [10] and MATA [11]. In particular, MATA was extended to support this work. The techniques have been applied to two realistic case studies. The first is a commercial electronic voting system, previously described by Kohno et al. [12]. The second is a positive train control system (PTC), previously considered by the second and third authors. PTCs are integrated command and control systems for ensuring train safety and are planned for deployment by the Federal Railroad Administration. We modeled security concerns for both applications using our technique. We believe that these two real-world case studies provide evidence for the practicability of the technique presented in the paper.

The paper is organized as follows. Section 2 introduces the key ideas using the electronic voting system as an illustrative example. Section 3 describes the technical contributions of the paper. Section 4 presents detailed results of the two case studies and is followed by related work and conclusions.

2. ILLUSTRATIVE EXAMPLE

Kohno et al [12] describe a security analysis of Diebold’s AccuVote-TS Electronic Voting System (EVS). This is a real EVS for which Kohno et al found significant security vulnerabilities. We use the EVS in this paper both as an introductory example but also as a validation of our work. This section presents a small part of the example.

Diebold’s EVS works as follows. An EVS allows voters to cast ballots at an electronic voting booth within a polling station. Two high-level business goals for an EVS are: (1) voter anonymity should be maintained, and (2) votes should be captured correctly. There are three principal use cases: (1) ballot definitions (i.e., details of candidates, party affiliations etc.) are sent to all voting machines at all polling stations, (2) voters cast ballots, (3) when voting is complete, the EVS reports the results. Voting is done using a smartcard. A voter registers at a polling station by showing his/her ID and is given a smartcard and PIN. S/he then proceeds to the EVS, enters the smartcard and PIN, and casts a vote.

Kohno et al. [12] list eleven attacks that could be perpetrated on Diebold’s EVS (based on an analysis of its source code). A simple example of a successful attack would be if an attacker were to manufacture his/her own smartcards that could then be used to cast multiple votes for a single voter, thus violating the second business goal above. We show in the remainder of this section how this attack and a possible mitigation can be modeled as executable misuse cases.

The behavior of the EVS must first be modeled as an extended UML interaction overview diagram (EIOD). EIODs will be explained in Section 3.1. We recommend a three level approach to modeling with EIODs. The top-level shows the use cases (Figure 1(a)). Each use case is then (optionally) described by another EIOD (Figures 1(b) and 1(c)) and finally, each of these IODs is described by a UML sequence diagram (see Figure 2). The use case model given by Figures 1 and 2 can be input to the UCSIM tool, which will automatically generate a set of FSMs, one for each participant in the sequence diagrams. Animation of these FSMs (available in UCSIM) can be used in validating the behavior of the EVS. FSMs generated for the voting machine, the smartcard and the voter are given in Figure 3.

The attack mentioned above can be captured as a misuse case. In our approach, a misuse case is also modeled by scenarios (i.e., sequences). Figure 4(a) shows how the attack relates to the voting use case. We use standard misuse case relationships. A misuse case <<threatens>> a use case if it potentially could prevent the use case’s goal from being achieved. A mitigating case <<mitigates>> a misuse case if it reduces the possibility of the attack being successful.
This attack succeeds because the FSMs do not prevent a second (manufactured) smartcard being inserted to cast an identical second vote. Once the modeler has modeled a successful attack scenario, s/he should design a mitigation strategy. This is done by defining mitigation scenarios—see Figure 4(c). A mitigation scenario is a sequence of events that will prevent the attack or reduce the chances of it being successful. In our approach, a mitigation scenario is an aspect scenario and defines mitigation messages that are woven automatically into the core behavior. This has the advantage of keeping the mitigations separate from the core behavior so that they can easily be modified later or reused. It also allows stakeholders to easily trade-off different mitigation strategies since it is easy to remove one mitigation strategy and replace it with another.

Figure 4(c) defines two mitigation scenarios for this first attack. In the original definition of the use cases, no voter-specific data was encoded on the voter smartcards. By encoding a unique ID for each voter, the EVS can avoid multiple votes from the same voter by recording the ID when a vote is cast and then checking for previous votes from the same ID when a new vote is cast. Note that only the voter ID, not the vote itself, is stored and so voter anonymity is preserved. Storing a voter ID on the smartcard will thwart a naive attacker that manufactures multiple faked smart cards but does not encode different IDs on each.

Storing a voter ID on the smartcard is specified on the left side of Figure 4(c). The right side specifies how to check for an existing ID. If one is found, there is a fraud attempt, and so a poll official is notified. The notation used will be explained fully in Section 3.2. For now, it is enough to know that any model element stereotyped with <<create>> will be added to the core behavior. Any element without a stereotype is matched against model elements in the core behavioral model. For example, the scenario on the left side of Figure 4(c) looks for a sequence of messages in the base where presentID is followed by authenticate. If found, a new message, storeVoterID, is added and a new smart card instance is also added. Similarly, on the right side of Figure 4(c), six new messages are added—five after insertSmartcard and one after submit. A new object, VoterData, is also added. The any fragment is used to match against a sequence of messages of undefined length. Hence, Figure 4(c) matches against any sequence of messages beginning with insertSmartcard and ending with submit, i.e., a sequence corresponding to a cast vote.

The MATA tool [11] was extended to automatically weave mitigation scenarios into a set of core behavior scenarios. The UCSIM tool then automatically generates a new set of FSMs that include both the original use case behavior and the new mitigation behavior. The attack scenario from Figure 4(b) can then be re-executed on the new set of FSMs to see if the attack still succeeds. In this case, it will not succeed because the check for an existing ID will stop an attempt at a second vote.

To run the attack scenario on the new set of FSMs, the FSM execution tool must be able to interpret the events existingID and NoExistingID. This is because the new FSM for the voting machine contains a branch after the smart card has been inserted. Two transitions form this branch—one for the case when there is no existing ID and one when there is. To execute the FSM any further, the execution tool must decide which branch...
to take. This can be resolved in one of two ways. Either the user directs the execution tool by injecting events. This allows the user to see what happens in both cases. Alternatively, the user can add new message specifications that provide interpretations that the execution tool can use. In this case, the user would add:

```
context storeVoterID context existingID
post: stored = true pre: stored = true
context NoExistingID
pre: stored = false
```

There are two points to note about this process. Firstly, executing the attacks and mitigations may lead project stakeholders to realize that whilst they have thwarted this particular attack, a more sophisticated attack (for example, where the attacker generates new IDs for each forged smartcard) may still succeed. This might in turn lead to a new mitigation. Secondly, a new mitigation may allow a previous attack to succeed because it nullifies a previous mitigation strategy. This situation can be easily detected by running all previous attack scenarios as a set of regression tests. Hence, this process can aid stakeholders in iteratively refining mitigation strategies.

3. TECHNICAL CONTRIBUTIONS

This section describes the key contributions of the paper. As stated in Section 1, one contribution is the integration of previous work on aspect-oriented scenarios and FSM synthesis. In addition, EIODs have been extended to support misuse cases (Section 3.1) and we have extended the expressiveness of our scenario weaving mechanism (Section 3.2).

3.1 Misuse Interactions

Extended interaction overview diagrams (EIODs), introduced in [4], are an extended form of UML [13] interaction overview diagram with a formally defined semantics [5], that model use cases at three levels. Level 1 is an extended activity diagram that shows use cases and their relationships. Each level 1 node (i.e., a use case) is refined at level 2 as an extended activity diagram. At level 2, each node is a scenario and the activity diagram therefore shows scenario relationships. Each level 2 node is refined at level 3 as a UML sequence diagram. An EIOD gives a complete, formally defined, description of a set of use case scenarios and the algorithm from [9] can be used to generate executable hierarchical FSMs automatically.

The relationships in use case charts go beyond those available in UML interaction overview diagrams. In particular, a scenario (or use case) may preempt or suspend another scenario (or use case). Scenarios (use cases) may have multiple concurrent executions. Scenarios (or use cases) may also be marked as failure cases, meaning that they stop execution. Finally, there is a well-defined notion of negative scenario in EIODs.

The details of EIODs are not crucial to this paper and a full set of definitions can be found in [5]. For security modeling, however, we extend EIODs to include misuse cases and misuse scenarios. All the EIOD relationships (preemption, suspension, concurrent executions etc.) are also available for misuse modeling. We call such a diagram a misuse EIOD. Misuse EIODs introduce two new relationships—<<threatens>> and <<mitigates>> (following Alexander [1]). In a misuse EIOD, a misuse case is modeled as an activity node at level 1 and is stereotyped as <<misuse>>. A mitigation case is modeled as a level 1 node with the stereotype <<mitigation>>. A level 1 misuse can be refined at level 2 by a set of related misuse scenario nodes, each stereotyped with <<misuse>>. Similarly, the refinement of a mitigation at level 1 is by another set of <<mitigation>> nodes at level 2.

At level 3, a misuse scenario is formed of two parts. The first part, called the modification scenario, describes how an attacker modifies the system under development in order for his/her attack to succeed. The intuition is that sometimes an attacker may be able to change the core behavior of the system. For example, an attacker might tamper with a smartcard so that it ignores certain commands. A modification scenario is optional for a given attack. If it exists, the modification scenario will be modeled as an aspect sequence diagram. The second part of a misuse scenario is the attack scenario, which describes the malicious interactions of the attacker with the system. Attack scenarios are given as standard UML sequence diagrams except that attackers are stereotyped with <<attacker>>. There may be multiple attackers.

Level 2 mitigations are defined at level 3 by a mitigation scenario, described as an aspect sequence diagram.

The example in Section 2 had no modification scenarios but had one attack scenario (Figure 4(b)) and two mitigation scenarios (Figure 4(c)).

Levels 1 and 2 of a misuse EIOD may employ relationships from EIODs as well as <<threatens>> and <<mitigates>>. A <<threatens>> edge has a <<misuse>> node as its source and a use case node (or scenario node) as its target. A <<mitigates>> edge has a <<mitigation>> node as its source and a <<misuse>> node as its target.

The semantics of an EIOD is given by the set of event traces that it admits (see [5]). For misuse EIODs, the semantics is given by translating a misuse EIOD into the equivalent EIOD. This is done by weaving the aspect sequence diagrams (which can be modification scenarios or mitigation scenarios) with the core behavior scenarios. We describe this translation procedure for a level 2 misuse EIOD. The procedure can be extended to level 1 misuse EIODs in the natural way.

A level 2 misuse EIOD is a triple \((N,E,s)\) where \((N,E)\) is a directed graph of nodes and edges, and \(s\) is a function mapping each node onto the set of level 3 sequences defining the node. The nodes \(N\) are partitioned into use case nodes, misuse nodes and mitigation nodes, \(N = U \cup M \cup M_t\). An edge is a triple \((n_1,n_2,l)\) between two nodes, labeled by its edge type \(l\). An edge \((n_2,n_2,thr)\) where \(n_1 \in M_s\), \(n_2 \in U\) is a <<threatens>> edge. An edge \((n_1, n_2, mit)\) where \(n_1 \in M_t\), \(n_2 \in M_s\) is a <<mitigates>> edge. For \(n \in M_s\), \(s(n)\) is partitioned into the set of modification scenarios \(s_{mod}(n)\) and the set of attack scenarios \(s_{att}(n)\).

For a misuse node, \(n \in M_t\), we say that a level 3 sequence diagram, \(d\), is in the scope of \(n\) if and only if there is a <<threatens>> edge from \(n\) to a node \(m \in U\) and \(d \in s(m)\). Similarly, for a mitigation node, \(n \in M_s\), we say that a level 3 sequence diagram, \(d\), is in the scope of \(n\) if and only if there is a <<mitigates>> edge from \(n\) to a misuse node, \(m \in M_s\), and \(d\) is in the scope of \(m\). Denote the scope of a node, \(n\), by \(\sigma(n)\).

To derive a level 2 EIOD from a level 2 misuse EIOD, firstly, for each misuse node, \(n \in M_s\), weave each modification...
scenario, \( s \in s_{\text{mod}}(n) \), with each sequence diagram \( d \in \sigma(n) \) and replace each \( d \) with \( \text{weave}(d,s) \) in the misuse EIOD, where \( \text{weave}(d,s) \) is the result of weaving \( s \) into \( d \). Delete all modification scenarios. This step modifies the core behavior of the system according to the modification scenarios.

Secondly, for each mitigation node, \( n \in M_r \), weave each mitigating scenario, \( s \in s(n) \), into each sequence diagram, \( d \in \sigma(n) \), in the scope of \( n \) and replace each \( d \) with the weave sequence diagram, \( \text{weave}(d,s) \). This step weaves the mitigating behaviors into the core behavior of the system. If all misuse and mitigation nodes are removed from the result, then we have an EIOD equivalent to the original misuse EIOD. The definition of weaving an aspect scenario into a sequence diagram is given in the next subsection. Note, however, that, in general, when there are multiple weaving steps, the order of weaving matters. Hence, the user must ultimately specify a weaving order. As described in [14], however, analysis techniques are available in MATA for determining ordering dependencies automatically.

Since modification and mitigation scenarios are given as aspect scenarios, they are crosscutting—in the sense that they cut across multiple core behavioral scenarios. Note that, according to the above translation procedure, the \(<\text{threatens}>\) and \(<\text{mitigates}>\) edges define the scope of this crosscutting. That is, they define the set of core scenarios crosscut by a scenario. For example, to find the set of sequence diagrams crosscut by a mitigating scenario, one traverses the misuse EIOD from the mitigation node across the \(<\text{mitigates}>\) and \(<\text{threatens}>\) edges to a use case node, which is, in turn, defined by a set of sequence diagrams. These sequence diagrams are those that are crosscut by the mitigation scenario.

### 3.2 Modeling and Weaving Aspect Scenarios

Mitigations and modifications are modeled as aspect scenarios. This is done to promote reusability of security-specific concerns, which tend to crosscut core functional concerns. An aspect scenario is a scenario that crosscuts other scenarios. We model aspect scenarios using the aspect modeling language MATA [11]. In MATA, a base sequence diagram is crosscut by an aspect sequence diagram. The base is simply a UML sequence diagram. The aspect sequence diagram is written using a MATA profile for UML that includes the following stereotypes: \(<\text{create}>\), \(<\text{delete}>\) and \(<\text{context}>\).

![Figure 5: Composed Mitigation and Base Scenario.](image)

![Figure 6: Sequence Pointcuts in MATA.](image)
MATA has been extended to include matching of using AGG as the back-end graph rule execution engine. Against but that are not needed by the aspect, by using the new messages in the base that are relevant to the aspect. Furthermore, By using sequence pointcuts, we can concisely capture all the straightforward manner since they are just another application of diagram. In other words, the aspect will only apply if a security concerns, however, as can be seen by the matching of fragment must be added around existing base elements. In particular, note in Figure 4(c), that the second mitigation must insert an alt fragment so that an existing message (submit) is placed inside the second operand of the fragment. Also, a new Voter Data instance must be created. Weaving strategies such as these are impossible to specify simply by applying before, after and around advices to sequence diagrams.

MATA takes a unified approach to model weaving in that the same specification mechanism can be used for any modeling language, as long as it has a well-defined metamodel. In particular, weaving for UML class diagrams, UML sequence diagrams and UML state diagrams is done in the same way—the modeler uses pattern matching on the LHS of a graph rule to specify join points and specifies aspect behavior on the RHS using <<create>> and <<delete>>.

MATA aspect weaving is based on graph transformations and is thus easily automatable as well as amenable to formal analysis. Full details can be found in [11]. Briefly, the MATA tool takes a base model, given in UML, and translates it into a graph. Similarly, a MATA aspect is translated into a graph rule. The graph rule execution tool AGG [15] is used to execute these graph rules (i.e., weave the aspects). AGG is based on the theory of attributed typed graphs and a type graph of the source and target model defines the abstract syntax of graphs that can be transformed. This type graph is obtained in MATA by translating the UML metamodels into the type graph form recognized by AGG.

For modeling security concerns as aspects, we extended MATA to include sequence pointcuts. A sequence pointcut is a sequence of messages to be matched against in the base sequence diagram. In other words, the aspect will only apply if a collection of messages are found to occur in a particular order. Most AOM approaches for sequence diagrams allow only a single message to be a join point. It is therefore impossible to take into account the fact that messages may be causally related. This turns out to be an extremely useful feature for modeling security concerns, however, as can be seen by the matching of both insertSmartCard and submit in Figure 4(c).

By using sequence pointcuts, we can concisely capture all the messages in the base that are relevant to the aspect. Furthermore, we can abstract away from messages that must be matched against but that are not needed by the aspect, by using the new MATA any fragment. An any fragment will match against any number of messages. Since MATA is based on graph transformations, sequence pointcuts can be handled in a straightforward manner since they are just another application of pattern matching-based weaving. The tool implementation of MATA has been extended to include matching of any fragments using AGG as the back-end graph rule execution engine.

Rule R2 in Figure 6(b) is a slight variation of rule R1 that uses a sequence pointcut to match against any sequence of messages that starts with p and ends with b. The messages occurring between p and b are unimportant to the aspect and so are matched against using the any fragment. (Note that since any fragments cannot be created by an aspect, they automatically are stereotyped with <<context>> if they occur inside a container stereotyped with <<create>> or <<delete>>.) The effect of applying R2 is different from R1 in that the messages q and b both appear inside the par fragment in the result—see Figure 6(c).

Sequence pointcuts turn out to be very convenient for modeling mitigation and modification scenarios as aspects. For the mitigation aspect in Figure 4(c), note how a sequence pointcut is used to match any sequence of messages coming after insertSmartCard and before submit. Sequence pointcuts allow all messages related to the mitigation aspect (in this case, storage of the voter ID) to be modeled on the same diagram.

### 3.3 Executing and Testing Misuse Cases

The previous subsection described how modification and mitigation scenarios are woven into the core use case model to produce an EIOD that combines both the core behavior and the mitigation behavior. Since EIODs have a formal semantics, this resulting EIOD can now be executed, animated or analyzed. Currently, our tool supports animation but not formal analysis. However, off-the-shelf analyzers for FSMs could be used easily.

Recall that the attack scenario behavior is not included in the composed EIOD. This is because attack scenarios describe an attacker’s behavior and therefore, its interactions do not form part of the system description. Instead, the attack scenarios are used as tests that can be automatically executed on the EIOD. If the tests fail, the EIOD is able to prevent the attacks. Otherwise, the mitigation strategies are not behaving as expected. Figure 7 outlines a recommended process for developing and testing executable misuse cases using the attack scenarios as a set of regression tests.

Executing the attack scenarios on the composed EIOD (step 3) is straightforward. Each attack scenario (i.e., UML sequence diagram) defines a set of event traces, so an attack scenario succeeds on an EIOD if its trace set is a subset of the traces described by the EIOD. Checking this, in practice, requires injecting events from the attack scenario into the EIOD animator.

Generally, events in the EIOD animator are uninterpreted. This can lead to nondeterminism when executing an EIOD, however. We saw an example of this in Section 2 where the animator needed to choose between the two events existingID and
The modeler may provide pre/post-condition specifications for messages that define the semantics of the messages. The animator can then check which preconditions are enabled to choose between events, thus resolving any nondeterminism. In practice, we have found that it is enough to write pre/post-conditions over global Boolean valued variables.

Modifications at the factory, for example, the inclusion of back disclosures. Finally, distribution attacks focus on malicious well as non-malicious insider actions that result in undesired disclosures. Finally, distribution attacks focus on malicious modifications at the factory, for example, the inclusion of back doors into software.

4. EVALUATION

4.1 Methodology

In this section, we report on two case studies undertaken to evaluate the procedure of Figure 7. The models for the first case study, the EVS from Section 2, were developed by the authors but based on the attacks and system behavior described in [12]. Models for the second case study were developed by the third author. This second study concerns a positive train control system (PTC), an intelligent control system currently planned by the Federal Railroad Administration. The case studies validate the executability of the modeling language. Further studies are necessary to validate the benefits of the technique in real teaming security requirements. The main result of this preliminary validation is that misuse EIODs are rich enough to express a wide range of realistic attacks and mitigations for real-world systems. In addition, the studies provide evidence that it is possible to animate precisely specified misuses in practice.

To quantify the range of attacks supported by our approach, we categorize attacks in each study according to two published classification schemes. A detailed taxonomy of attack types is presented in [17]. We limit ourselves to the portion of this taxonomy that catalogs attacks according to the result of the attack. In [17], there are five possible results: corruption of information, disclosure of information, denial of service, increased access and theft of resources. The second classification scheme is from the Information Assurance Technical Framework Forum (IATFF) [18] (sec. 1.3.5), which classifies attacks based on where the attacks occur. Attacks can be categorized as passive, active, close-in, insider, and distribution. Passive attacks come from external attackers who monitor and analyze system characteristics (e.g., network traffic) in order to obtain sensitive information. Active attacks come from external attackers who not only monitor a system but

Figure 8: Attack 2—Deactivate Disable Card Message.

4.2 EVS Results

We continue with the EVS example and demonstrate how to model an attack that includes a modification scenario. We then report on the overall results for the EVS system.

The second attack mentioned in [12] is a variation of the attack considered previously. Voting machines disable smartcards once a voter submits his/her vote. Therefore, if an attacker is able to reprogram a smartcard to ignore the disable message, s/he could reuse the card to cast multiple votes. Figure 8 shows the models for this attack. The misuse consists of both a modification scenario and an attack scenario. The modification scenario captures what an attacker must do to the EVS for the attack to succeed. S/he must modify the smartcard so that it ignores the disableCard message (see Figure 2). Figure 3(b) gives the FSM defining the behavior for smartcards. Therefore, for this modification, the FSM must be changed by the attacker. To model this, the modeler need only write the modification scenario, which will be weaved into the smartcard FSM to reflect the behavior of the smartcard once the malicious action of ignoring the disableCard message has been taken. The modifications are modeled using aspects so that they can be maintained separately from the rest of the model.

Figure 8(b) is the modification scenario. It looks for a pattern containing a disableCard directive sent from the voting machine to the smartcard. It then removes the action that disables the smartcard. This is done using MATA’s "<<delete>>" stereotype which removes a model element from a base model. The attack scenario is the same as for attack 1 but is included in Figure 8 for completeness. Note again that it is of no concern to the model how the attacker manages to carry out this modification—such considerations are outside the scope of the model. Once the modification scenario is weaved into the core
behavior model, the weaved model captures the preparations that the attacker has made so that his/her attack can succeed. Note that there is a check in Figure 2(c) to see if the smartcard is enabled or not: a message checkEnabled is sent from the voting machine to the smartcard. For animation of the weaved FSM, the response of the smartcard should be linked with the disableCard action. As before, this can be done by adding a precondition of enabled=true to checkEnabled and a postcondition of enabled=false to disableCard.

No additional mitigation scenario is necessary for this particular attack since the mitigation for the first attack will also thwart this one. This is because the ID on the smartcard is unchanged when the attacker tries to reuse it, and so the check for an existing ID will still work. Note that the automation provided by executable modeling can handle this automatically. In step 3 of the process in Figure 7, the modeler would execute the two attacks and find that both are mitigated. Hence, there is no need for a second mitigation scenario. Of course, with only two attacks, it is fairly straightforward to realize this without the benefit of automation. As the number of attacks increases, however, the benefits of automation become apparent.

Table 1 summarizes the attacks that were modeled for the EVS system. [12] discusses attacks arising both from design flaws and coding flaws. Only the design flaws are relevant for misuse modeling. In total, there were nine such attacks.

### Table 1: EVS Attacks.

<table>
<thead>
<tr>
<th>ID</th>
<th>Attack Name</th>
<th>Attack Category 1 [17]</th>
<th>Attack Category 2 [18]</th>
<th>Can be modeled?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Manufacture own smartcards</td>
<td>Corruption of Information</td>
<td>Active/Close-in</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>Deactivate smartcard</td>
<td>Corruption of Information</td>
<td>Active/Close-in</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>Close voting prematurely</td>
<td>Denial of service</td>
<td>Active/Close-in</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>Modify ballot definition</td>
<td>Corruption of Information</td>
<td>Active</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>Impersonate voting machine</td>
<td>Corruption of Information</td>
<td>Active/Insider</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>Tamper with election results</td>
<td>Corruption of Information</td>
<td>Active</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>Link voters with votes</td>
<td>Disclosure of Information</td>
<td>Passive/Insider</td>
<td>N</td>
</tr>
<tr>
<td>8</td>
<td>Tamper with audit logs</td>
<td>Corruption of Information</td>
<td>Active/Insider</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>Delay start of election</td>
<td>Denial of Service</td>
<td>Active</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 1 shows a fairly wide range of attacks that can be modeled. As can be seen from the table, the attacks fall into three of the five categories for classification 1, and for classification 2, only distribution attacks are not covered.

Attack 7 could not be modeled because the attack only becomes apparent during low-level design since it concerns how data is stored. Attack 9 is a network-based attack. One would not expect a requirements modeling technique to be able to capture these attacks. Rather, these attacks would manifest themselves during detailed design, so it is desirable to continue the executable modeling of security concerns to later stages of the development lifecycle.

### 4.3 PTC Results

The second study modeled attacks for positive train control systems (PTCs), which are planned for deployment by the Federal Railroad Administration. PTCs [19] are distributed, interoperable wireless communications-based railroad control systems whose primary use cases aim to: (1) prevent train-to-train collisions, (2) enforce train speed restrictions, and (3) protect roadway workers. The functionality of a typical PTC was modeled by an EIOD that included five level 1 use cases. In a brainstorming session, a total of 19 attacks on these use cases were developed at a very high-level. It was realized that, of these 19 attacks, many were similar. Therefore, instead of modeling each attack individually, four representative attacks were chosen to be modeled. Models for the other 15 could easily be obtained, if desired, by modifying one of these four samples.

### Table 2: PTC Attacks.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Modify Track Warrant</td>
<td>Corruption of Information</td>
<td>Active</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>Jam Wayside Unit</td>
<td>Denial of service</td>
<td>Active</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>Power Exhaustion Attack</td>
<td>Denial of service</td>
<td>Active</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>Generic Insider attack</td>
<td>Corruption of Information/Increased Access</td>
<td>Insider</td>
<td>Y</td>
</tr>
</tbody>
</table>

We briefly describe these four attacks here and summarize them in Table 2. PTCs rely on successful interactions between three classes of actors: the central dispatcher that directs railroad operations, mobile units (i.e., locomotives) that transport passengers and freight, and wayside units (e.g., signals). Though the individual subsystems in a PTC system are fail safe, the reliance of PTCs on wireless networks invites exploits that, unless explicitly addressed, may endanger rail operations. The first attack is a classic man-in-the-middle attack in which track warrants (authorizations issued from the dispatcher to mobile units) are intercepted by an attacker and modified en-route, thus potentially leading to collisions. In the misuse EIOD, this attack was modeled using a modification scenario that inserted messages into the core EIOD. These messages changed the data transmitted by a sendMovementAuthority message and also added messages so that mobile units would react adversely to the modified data. Additional authentication procedures were modeled aspect mitigation scenarios.

The second and third attacks were types of denial of service attacks—the second involved jamming wayside units and the third involved a power exhaustion attack on a wayside unit. Both of these could be modeled using a modification scenario to mimic the inability of wayside units to respond when under attack. A couple of mitigation scenarios were modeled. One involved the detection of the lack of responsiveness and subsequent rerouting of messages. The other introduced privileged frequencies for transmissions.

The fourth attack involved an insider at the dispatch office transmitting track warrants that caused collisions. The aspect mitigation scenario was to introduce a mediating authority. Table 2 classifies these four attacks.
4.4 Discussion
The two case studies are very different in nature. The PTC vulnerabilities arise mainly from the use of wireless communications. For the EVS, on the other hand, attacks commonly involve system access via modification of a physical resource (i.e., a smartcard).

Misuse EIODs tend to be most effective for modeling application-level security concerns. Network-level concerns can often be handled by standard encryption techniques and there seems to be limited benefit in modeling these precisely at the requirements level. Encryption could be handled in misuse EIODs, however, and it may be beneficial to maintain a library of misuses, such as network concerns, that can easily be adapted to a particular system. Misuse EIODs are well suited for modeling insider attacks. These attacks cannot be prevented by using standard encryption and require more complex mitigations such as monitoring and systems that periodically remove orphaned accounts. Misuse EIODs are also particularly well suited for modeling application-level or system-level attacks and could effectively be used, for example, to model more traditional, non-cyber security concerns, such as airport security.

Currently, misuse EIODs are a rather general modeling paradigm. This is both a strength and a weakness. It admits a wide range of attack types to be modeled. For security, however, there is a vast amount of domain knowledge available that could be brought to bear. For example, well known threat modeling methodologies (e.g., [20, 21]) advocate starting with generic versions of well known attacks, such as attack patterns. Incorporating attack patterns into misuse EIODs would improve the efficiency of misuse EIOD modeling. Right now, as with any modeling technique, a reasonable degree of expertise and experience is required to translate a high-level description of an attack into a precise misuse model. We believe, therefore, that a valuable addition to misuse EIODs would be to formalize well known attacks as generic patterns that could be instantiated to a specific application. In fact, it appears that aspects would be ideal for this purpose.

5. RELATED WORK
Misuse cases were first introduced by Sindre and Opdahl [2] and were developed further by Alexander [1]. Similarly, abuse cases [22] also consider a system from the viewpoint of what should be prevented. Misuse (and abuse) cases have received widespread attention because of their similarity to use cases and their intuitive nature. Despite this, support for misuse cases has been limited to defining the role of misuse cases in a software process [21], templates for describing misuse cases textually [23], and relatively simple tool support such as Alexander’s ScenarioPlus [24]. These approaches succeed in promoting the utility of misuse cases but do so in a chiefly informal way. To the authors’ knowledge, misuse EIODs are the first executable language for modeling with misuses.

The software industry has risen to the challenge of secure software engineering and there are a number of processes that attempt to address security concerns early in the development lifecycle. The most well known perhaps is Microsoft’s threat modeling approach [20]. Others include the Security Quality Requirements (SQUARE) Methodology [25]. These are largely informal processes focusing on best practices and systematic guidelines. Misuse EIODs, however, are executable but could be used as a part of such processes.

Within requirements engineering, security has been considered by many existing requirements engineering methods. In [26], van Lamsweerde extends KAOS for modeling application-specific security requirements. Templates specify common security goals, where the goal is specified in first order logic using specialized predicates such as authorized, knows etc. Threats are modeled as (fault) trees, referred to as obstacle trees, in which the root is a negation of a security goal. The paper uses these anti-goals to derive objects and how attackers may use them in attacking the system under design.

Secure Tropos [27] is an agent-based design methodology for designing secure systems. It uses actors, roles, goals, tasks, resources, objectives, capabilities delegation and trust as basic design concepts. A goal represents strategic interests of the actor, and a task represents a course of action taken to satisfy a goal. The system uses permissions, delegating permissions based on trust etc. The main focus is on trust rather than adversary behavior.

Liu et al. [28] describe a social-science oriented requirements analysis method for secure systems. In this method, (a precursor to Secure Tropos), actors and attackers and their goals are identified. Then goals are refined to tasks, and their dependencies are specified. Mal-actor intent, and vulnerabilities are also identified, and their dependencies are used to derive the effectiveness of proposed counter measures. These concepts are formalized using Alloy.

Goal-based methods such as KAOS and Secure Tropos have not achieved as widespread use across industry as use cases. Therefore, we believe the two to be complementary. In fact, goals and scenario-based approaches can be effectively used together [29]. Goal models provide a systematic process for considering security goals and anti-goals at a high-level without showing detailed sequencing between goals. This sequencing can be provided by use and misuse cases. Also, since our misuse cases are executable, they permit automated red-teaming. Although some kinds of analysis can be done with goal models, the analysis often requires formal notations which are unlikely to be used by practitioners. It is an open question how best these two kinds of techniques can be used together. Note also that attack trees [30], which are used widely by industry, can be considered as a special case of goal models.

There has been a good deal of work on using aspects to represent security concerns. This has been chiefly at the design level, however, and so is complementary to the work in this paper. Kim uses a role-based metamodeling language to capture access control models [31] as aspects. Song et al [32] take a similar approach but consider security concerns as aspect sequence diagrams.

6. CONCLUSION
This paper presented a new technique for precisely modeling and executing misuse case scenarios for secure systems development. The work integrates previous research on executable modeling with scenarios and on weaving aspect scenarios. However, the paper makes contributions to both of these areas as well as applying the integration of the two techniques to a new application area. The result is a modeling approach that allows stakeholders to brainstorm potential security attacks, capture mitigations of those attacks, and test whether the attacks have indeed been thwarted.
Future work will investigate how to transform the results of analysis into later design stages. We will examine how to apply a model-driven approach to develop secure designs in such a way that the separation of core behavior and mitigation behavior, which was achieved at the requirements level, can be maintained during design. We also plan to understand better the exact relationship between goal-based models, such as attack trees, and misuse case models. Finally, we will investigate how to support misuse case modeling, for example, by generating attack sequences automatically.

7. ACKNOWLEDGMENTS
Thanks to Amr Said who implemented MUCSIM.

8. REFERENCES