Suspicion-driven formal analysis of security requirements

Nuno Amálio
University of Luxembourg, 6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
nuno.amalio@uni.lu

Abstract

Increasingly, engineers need to approach security and software engineering in a unified way. This paper presents an approach to the formal analysis of security requirements that is based on planning and uses the concept of suspicion to guide the search for threats and security vulnerabilities in requirements. The approach is tested and illustrated by conducting two experiments: one focussing on a system with a confidentiality security property, and another with an integrity security property enforced through the separation of duty principle. The paper shows that suspicion plays an important rôle in finding vulnerabilities and security threats in requirements.

Keywords: *Security, requirements, formal analysis, Event-Calculus, planning, confidentiality, separation of duty.*

1. Introduction

Traditional approaches to software engineering and current practice tend to treat security concerns as an afterthought [1]. Security requirements are handled as non-functional requirements and are kept separate from their functional counter-parts until design or implementation-time. This raises problems for the whole software development process because (as demonstrated in this paper) functionality has an impact on security. If security aspects are not treated properly at the requirements phase, then the resolution of the problem will inevitably be deferred but at a much higher cost, which is a well known software engineering problem [2]. This issue can be resolved by integrating security into the requirements engineering phase of the software life-cycle [1], [3]. However, the best way to capture, model and analyse security and system requirements in a unified way is still an open problem.

The techniques developed to formally model and analyse general software systems are also applicable to model and analyse security. However, the emphasis of the analysis is different. System analysis focusses on *safety* (something will not happen) and *liveness* (something must happen) [4]; safety checks that bad states cannot be reached and liveness that good state are reached. This is used to check that invariants are preserved, that operations are applicable when certain conditions are met (pre-conditions) and that the operation has the desired effect taking the system into a valid state. In security, the emphasis is not only in verifying *safety* (that some unsecure state is not reached) and liveness (that the security measures do what is expected from them), but also in finding security vulnerabilities and possible security threats that give attack opportunities to malicious users: we need to look for *what can happen if some condition holds*.

This paper presents an approach to the formal analysis of security requirements based on the concept of suspicion. The approach consists of searching for security vulnerabilities and threats based on what is suspicious. This search is based on AI planning. From a formal model of requirements and an analysis goal (a description of interesting states from the analysis point of view), planning generates plans that reach the goal state. Each plan gives a scenario illustrating a possible threat or security vulnerability. The generation of plans can be done automatically with tool support. The approach is illustrated with the Event-Calculus temporal logic [5] and the analysis is conducted with tool support using the discrete event calculus reasoner[1] (*decreasoner*).

The remainder of this paper starts by discussing related work (section 2). Then, it gives a brief introduction to event calculus (section 3) and presents the approach to the formal analysis of security requirements that is proposed here (section 4). After this, the analysis approach is used to conduct two experiments; one focussing on the analysis of a system with a security requirement (section 5), and another to a business system with an integrity requirement enforcing separation of duty (section 5). Finally, the paper discusses the results and takes the conclusions.

2. Related Work

There has been substantial interest on security requirements threat analysis [6], [3], [7], [8]. In [7], [8], specifiers need to explicitly identify scenarios or use-cases of abuse and misuse; here, such scenarios are generated automatically from a description of suspicious states (the goal).

[6] proposes a method based on the more flexible abuse frames, specifying undesirable phenomena that the system should prevent from happening. The approach presented here enables the specification of such undesirable phenomena as goals (here called security violation goals). However, it does not only consider what should not happen, but also considers

1. http://decreasoner.sourceforge.net/

specification of flexible suspicious conditions that (as shown here) have the potential of exposing unnown threats.

[3] proposes a goal-based method that is similar to the approach presented here. Security goals, such as confidentiality, integrity and availability, are negated to obtain goals that specify what should not happen (our security violation goals). Then, these negations are refined to obtain more flexible goals. The approach presented here is more flexible in that it does not only allow specification of goals that come from negation and refinement security goals, but also leaves the specifier the flexibility of defining what constitutes a suspicious condition. Since it is based on tool support, the specifier can use the feedback coming from the tool to either refine analysis goals or specify new analysis goals. Essentially, the work presented here is complementary to the body of work on security requirements threat analysis. It explores automated formal analysis (missing in the works above) and provides experimental evidence to the usefulness and effectiveness of security threat analysis.

Suspicion is ubiquitous in *intrusion detection* [9]. Anomaly-based approaches to intrusion detection [9] are so called because the search for intrusions is driven by abnormal (or suspicious) behaviour patterns of system use. [10] proposes the inclusion of suspicion as a concept driving the models of *misuse-based* intrusion detection; it proposes models based on suspicious activities that may lead to an attack, as opposed to models based on actual attacks. Instead, the approach presented here detects abuse by identifying suspicious states in a model of normal system behaviour.

The approach presented here emerges from its preceding work on threat detection [11]. [11] uses an EC model of requirements and planning to find threats at run-time. In [11], however, the goals used to detect threats are more rigid than the ones used here; they say with absolute certainty whether there is an attack or not when the goal is satisfied. In the approach presented here, there is no certainty of attack if a suspicious goal is satisfied, the plans that reach the goal just give us threats (possible attacks). It would be possible to incorporate our approach based on suspicion as a strategy in looking for threats at run-time. Then, the probabilistic component of such a system (like the one of [11]) would try to assign a probability to the computed threats. Here we use suspicion to look for threats in requirements to avoid systems with security vulnerabilities.

The approach presented here analyses requirements automatically using a tool based on SAT-solving. The advantage of this method with respect to other approaches based on theorem-proving [12], [13] is that the reasoning is automatic, avoiding the need for user-intervation as it is usually the case with theorem proving. The disadvantages are that only a portion of the state space is analysed, and that the models that can be handled need to be small. Another disadvantage to the work in [12], [13] is that there is no visual description of requirements and properties to check; the user needs to be an expert in the formal language (here EC).

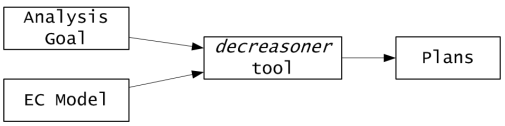

Figure 1. Planning-based formal analysis. EC model of requirements, and EC description of analysis goal are fed into *decreasoner* tool to obtain a set of plans (scenarios) that achieve the goal.

3. The Event Calculus

Event Calculus (EC) [5] is a language based on first-order predicate-calculus enabling representation and reasoning about action and change. Its basic ontology comprises *events*, *fluents* and *timepoints*. An *event* is an action that may occur in the world. A *fluent* is a time varying property of the world. A *timepoint* is an instance of time. EC includes a set of basic predicates to describe happening of events, their effects and state of fluents.

The basic predicates of EC are as follows:

- $HoldsAt\ (f,t)$ says that fluent f is true at timepoint t.
- $Happens\ (e,t)$ says that event e may occur at timepoint t.
- $Initiates\ (e,f,t)$ says that if event e occurs at timepoint t, then fluent f is true after t.
- $Terminates\ (e,f,t)$ says that if event e occurs at timepoint t, then fluent f is false after t.
- $Initially\ (f)$ says that fluent f holds at timepoint 0.

4. Formal threat analysis based on planning

AI planning [14] underpins the analysis approach presented here. The analysis is essentially a study of *reachability*: it checks whether certain states are reachable. The actual planning is conducted in the framework of satisfiability (SAT) problem solving using the *decresoner* tool, which is based on a SAT approach to EC reasoning [15].

Figure 1 depicts the analysis approach followed here. The EC model and EC description of analysis goals are given as inputs to *decreasoner*, which generates a set of plans that satisfy the goal. Each plan describes a scenario comprising a sequence of events (a *trace*) that take the system from the initial state to one of the states described by the goal.

The goal describes a set of states that are interesting from the analysis point of view. Planning generates plans that reach such a state. If there are plans, then the goal is satisfiable: a state as described by the goal can be reached in the model. If no plans can be found, then no state as described by the goal is reachable. If the goal state describes

R1	Doctors must be able to access their patient's medical data to provide effective medical care.
R2	A doctor may nominate a substitute doctor who may be able to access the patient's medical data only when the main doctor is on leave.

Table 1. The requirements that rule the access to patient's information in SMIS.

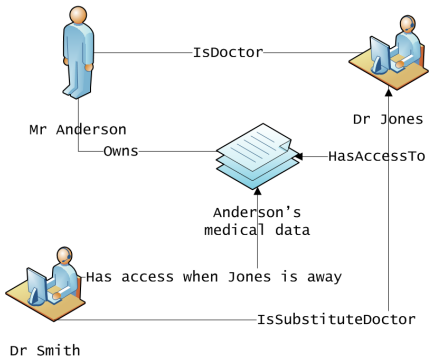

Figure 2. The SMIS with one patient, Anderson, his doctor Jones, and another doctor, Smith, who is able to replace Jones while he is on leave.

something that should not happen, then the resulting plans (scenarios) describe a sequence of events that reach such a state; thus exposing a way to reach something undesired.

Two strategies are used to formulate the goal. There is a more traditional strategy that does a *safety analysis* by formulating a goal describing states where security is violated and that must not happen; these goals are called *security violation* goals. The other strategy applies suspicion by defining a goal describing *suspicious states deserving investigation* that may expose possible vulnerabilities and threats. These are called *suspicious goals*.

5. Analysing confidentiality

Confidentiality is about protecting information. It tries to ensure that sensitive information is accessible only to those authorised to access it. Analysis of confidentiality involves checking ways in which confidential information may be accessed by those who are not authorised. Here, confidentiality is studied using a case study of a domain where it is a professional ethical principle: health-care [16].

The case study is a simple medical information system (SMIS) that manages patient information. Due to its sensitive nature, patient information is subject to confidentiality constraints to protect the patient's privacy. The requirements of SMIS are summarised in table 1. Figure 2 depicts a concrete system scenario of SMIS.

The EC model of SMIS requirements is given in figure 3. The model considers that doctors need to obtain credentials

$$\forall\, d : Doctor; p : Patient; t : Time \mid HoldsAt\ (CanAccessMD\ (d,p),t) \Leftrightarrow HoldsAt\ (IsDoctorOf\ (d,p),t) \vee ((\exists d' : Doctor)\ HoldsAt\ (IsDoctorOf\ (d',p),t) \wedge HoldsAt\ (IsSubstituteDoctor\ (d,d'),t) \wedge HoldsAt\ (OnLeave\ (d'),t)) \quad (1)$$

$$\forall\, d : Doctor; p : Patient; t : Time \mid HoldsAt\ (CanAccessMD\ (d,p),t) \Rightarrow Initiates\ (AuthoriseAccess\ (d,p), CredentialMD\ (d,p,t),t) \quad (2)$$

$$\forall\, d : Doctor; p : Patient; t : Time \mid HoldsAt\ (HasValidCredential\ (d,p),t) \Leftrightarrow (\exists t_2 : Time)\ HoldsAt\ (CredentialMD\ (d,p,t_2),t) \wedge (t_2 + 3) >= t \quad (3)$$

$$\forall\, d : Doctor; p : Patient; t : Time \mid Happens\ (GetMD(d,p),t) \Rightarrow HoldsAt\ (HasValidCredential\ (d,p),t) \quad (4)$$

$$\forall\, d : Doctor; p : Patient; t : Time \mid Initiates\ (GetMD\ (d,p), ExposedToAt\ (d,p,t),t) \quad (5)$$

$$\forall\, a : Agent; d_1, d_2 : Doctor; t : Time \mid Initiates\ (SetSubstituteDoctor\ (a,d_1,d_2), IsSubstituteDoctor\ (d_2,d_1),t) \quad (6)$$

$$\forall\, a : Agent; d : Doctor; t : Time \mid Initiates\ (SetDoctorOnLeave\ (a,d), OnLeave\ (d),t) \quad (7)$$

$$\forall\, a : Agent; d : Doctor; t : Time \mid Terminates\ (DoctorNoLongerOnLeave\ (a,d), OnLeave\ (d),t) \quad (8)$$

$$\forall\, d : Doctor; p : Patient; t : Time \mid Initially\ (\neg\ ExposedToAt\ (d,p,t)) \quad (9)$$

$$\forall\, d_1, d_2 : Doctor \mid Initially\ (\neg\ IsSubstituteDoctor\ (d_1,d_2)) \quad (10)$$

$$\forall\, d : Doctor \mid Initially\ (\neg\ OnLeave\ (d)) \quad (11)$$

$$Initially\ (IsDoctorOf\ (jones, anderson)) \quad (12)$$

Figure 3. Initial EC formalisation of the security requirements of the SMIS.

in order to access patient's data. Credentials are obtained through event *AuthoriseAccess*; patient's data is accessed through event $GetMD$. The system records substitute doctors through event $SetSubstituteDoctor$, is informed of doctors absences through event $SetDoctorOnLeave$, and of doctors return to duty after on leave through event $DoctorNoLongerOnLeave$.

Analysis uses the configuration depicted in figure 2. There are two doctors, Jones and Smith, and a patient of Jones, Anderson. Analysis starts by formulating a security violation goal: it asks whether it is possible to reach a state where patient confidentiality is compromised. In the context of SMIS, this happens when some doctor accesses some patient's data without a valid security credential; the goal expressing this is formulated as:

$$\exists\, d : Doctor; p : Patient; t_1, t_2 : Time \mid HoldsAt\ (ExposedToAt\ (d,p,t_2),t_1) \wedge \neg\ HoldsAt\ (HasValidCredential\ (d,p),t_2)$$

For this goal, *decreasoner* does not find any plans. This means that the modelled system cannot reach one of the goal's states. At this point, one could argue that the modelled system is secure because it is not possible to reach an unsecure state, but it isn't so.

Analysis proceeds by applying suspicion. The substitute doctor rule (Requirement $R2$) enables access to patient's

R2'	A substitute doctor may be nominated by the doctor that is to be substituted or by one of his administrators only. A substitute doctor may be able to see the medical data of patients of the doctor being substituted while he/she is on leave only.
R3	Only the doctor himself or one of his administrators may inform the SMIS of a doctors absence from duty (on leave) or that a doctor is back to duty.

Table 2. Requirements emerging from analysis to the SMIS early requirements.

$$\forall\ a : Agent; d : Doctor; t : Time \mid$$
$$HoldsAt\ (CanDoAdminForDoctor\ (a, d), t)$$
$$\Leftrightarrow a = d\ \vee\ (\exists\ ad : Admin \mid a = ad$$
$$\wedge\ HoldsAt\ (HasAdmin\ (d, ad), t)) \quad (13)$$
$$\forall\ a : Agent; d : Doctor; t : Time \mid$$
$$Happens\ (SetDoctorOnLeave\ (a, d), t)$$
$$\Rightarrow HoldsAt\ (CanDoAdminForDoctor\ (a, d), t) \quad (14)$$
$$\forall\ a : Agent; d : Doctor; t : Time \mid$$
$$Happens\ (DoctorNoLongerOnLeave(a, d), t)$$
$$\Rightarrow HoldsAt(CanDoAdminForDoctor(a, d), t) \quad (15)$$
$$Initially\ (HasAdmin\ (Jones, Alice)) \quad (16)$$
$$Initially\ (HasAdmin\ (Smith, Sue)) \quad (17)$$

Figure 4. EC formalisation of the security requirements of the SMIS after analysis uncovered problems with EC model of figure 3.

data by doctors other than the main patient's doctor. This should occur, but not very often; the situations under which this occurs are suspicious and deserve investigation. Analysis investigates states where those other than the main doctor access the patient's data. This is formulated as the goal:

$$\exists\ d : Doctor; p : Patient; t_1, t_2 : Time \mid$$
$$HoldsAt\ (ExposedToAt\ (d, p, t_2), t_1)$$
$$\wedge\neg\ HoldsAt\ (IsDoctorOf\ (d, p), t_2)$$

For this goal, *decreasoner* generates plans exposing a security vulnerability. In some scenarios, the system behaves as it should: $Jones$ sets $Smith$ as his substitute, at a later time $Jones$ informs system that he is on leave, and so $Smith$ is able to access the medical data. Other scenarios are more unusual. In some of them, it is possible that $Smith$ himself requests to be the substitute of $Jones$. In another, it is $Smith$ who informs the system that $Jones$ is on-leave. Obviously this is strange and could be explored by a malicious doctor determined to get some patient's medical data: (a) he set's himself as the substitute doctor, then (b) he informs the system that main doctor is on-leave and (c) finally he is able to access the patient's medical file.

Such scenarios are possible because events $IsSubstituteDoctor$ and $SetDoctorOnLeave$ (equations 6 and 7 in figure 3) are uconstrained: any agent may execute them and this introduces a loophole, providing an opportunity to access medical data in non-legal ways.

These analysis findings are used to elaborate the requirements. The requirements that emerge as a result of this

R1	There are two types of users clerks and managers. Managers can performs the tasks that usually the clerks do, but clerks should not usually perform manager's tasks (exception is delegation, below).
R2	Clerks are responsible for starting the refund procedure, and for issuing or cancelling the refund.
R3	The refund shall be issued by a clerk if approved by the managers, or cancelled otherwise.
R4	A refund must by approved by two different managers.
R5	A clerk shall not both prepare and issue or cancel a refund.
R6	Managers can delegate the authority on approval of refunds to one of their administrators.

Table 3. The requirements that rule the processing of tax refunds.

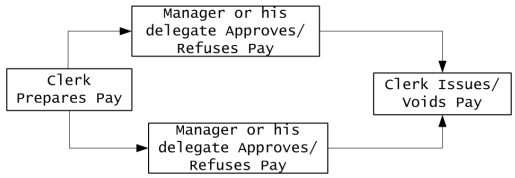

Figure 5. The payment processing workflow of the tax refund process.

clarification are given in table 2; here, $R1$ of table 1 still holds, $R2$ is replaced by $R2'$, and there is the new requirement $R3$ which introduces administrators. This results in the additional EC formulas given in figure 4.

Under these new requirements, we re-submit the model to the security violation and suspicion goals. For the security violation goal we still get no plans (not possible to break confidentiality in an obvious way). For the suspicious goal, we no longer get obvious security threats, but the results still prompt interesting requirements questions, such as: the system allows doctors to operate the system while they are recorded as being on leave, should this be allowed?

6. Analysing Separation of Duty

Separation of Duty (SoD) [17], [18] is a security mechanism used to prevent fraud and errors. It aims to prevent a single individual from executing sensitive tasks of some sensitive transaction. SoD requires such tasks to be performed by different users acting in cooperation (e.g by requiring two persons to sign a cheque). Here, SoD is studied using a classical case study: a workflow of payment processing, where SoD is put in place by requiring payment authorisations to be executed by different users.

The requirements of this worklflow system are given in table 3. The underlying workflow is depicted in figure 5. The two tasks involving approval of payments to be carried out by managers, and the tasks *prepare payment* and *issue/void payment*, to be carried out by clerks, are subject to SoD.

The EC model (not given here) specifies workflow, system

requirements, rôles and users. Workflow is modelled as a set of activities made of several alternative tasks; one of the tasks must be carried out in order to complete the activity. In the workflow of figure 5, activity *Approve/Refuse Pay* comprises tasks *approve pay* and *refuse pay*, and activity *Issue/Void Pay* comprises tasks *issue pay* and *void pay*.

Analysis is conducted in a configuration made of three managers, Bob, John and Martin, and three clerks Sam, Alice and Sue; Sue also works as an administrator for John. It starts with a security violation goal to know if it is possible to reach a state where SoD is breached. This happens if a user executes two different tasks belonging to activities constrained under SoD. This is expressed as the predicate:

$$\begin{aligned}&\forall\ u : User; s : Session; t : Time \mid \\ &\quad HoldsAt\ (BreaksSoD\ (u, s), t) \\ &\quad \Leftrightarrow \exists\ a_1, a_2 : Activity \mid a_1 \neq a_2 \\ &\qquad \wedge\ HoldsAt\ (ExecTaskOfActivity\ (u, a_1, s), t) \\ &\qquad \wedge\ HoldsAt\ (ExecTaskOfActivity\ (u, a_2, s), t) \\ &\qquad \wedge\ (SoD\ (a_1, a_2) \vee SoD\ (a_2, a_1))\end{aligned}$$

From this predicate, the goal is defined by describing states where SoD is breached:

$$\begin{aligned}&\exists u : User;\ s : Session;\ t : Time \mid \\ &\qquad HoldsAt\ (BreaksSoD\ (u, s), t)\end{aligned}$$

For this goal, *decreasoner* does not find any plans. This means that it is not possible to reach a state where SoD is breached. Again, one could argue that SoD is preserved and the system is secure, but it isn't so.

Analysis proceeds by investigating the suspicious. Although a user is allowed to execute more than one task in some workflow session, this should not happen very often and is suspicious. First, we define a predicate describing the suspicious system condition of having a user executing two tasks in a workflow session:

$$\begin{aligned}&\forall\ u : User; s : Session; t : Time \mid \\ &\quad HoldsAt\ (HasExecutedTwoTasks\ (u, s), t) \\ &\quad \Leftrightarrow \exists\ ta_1, ta_2 : Task; t_2, t_3 : Time \mid \\ &\qquad \wedge\ Happens\ (ExecTask\ (ta_1, u, s), t_2) \\ &\qquad \wedge\ Happens\ (ExecTask\ (ta_2, u, s), t_3) \\ &\qquad \wedge\ ta_1 \neq ta_2 \wedge t_2 \leq t_1 \wedge t_3 \leq t_1\end{aligned}$$

From this predicate, the suspicious goal describes states where some user executes two different tasks in some workflow run:

$$\begin{aligned}&\exists\ u : User; s : Session; t : Time \mid \\ &\quad HoldsAt\ (IsWrkfComplete\ (s), t) \\ &\quad \wedge HoldsAt\ (HasExecutedTwoTasks\ (u, s), t)\end{aligned}$$

For this goal, *decreasoner* generates interesting plans. We have scenarios where a manager prepares the payment and then approves it, or that he approves and then issues the payment. This happens because managers may act as clerks and there is no SoD constraint between tasks that managers do and clerks do; this may give a fraud opportunity. Clarification of this requirements results in new requirement $R7$ (table 4), which says that a person can execute tasks under at most one role in any workflow session. After the fix, this behaviour is no longer allowed.

R6'	Managers can delegate the authority on approval of refunds to one of their administrators, but when the administrator executes such task it should be considered that is as if it had been executed by the manager.
R7	The same person may perform tasks as either manager or clerk, but not both, in any workflow session.

Table 4. Requirements that rule the processing of tax refunds that emerged after analysis.

Analysis turns to delegation, which is known to generate security vulnerabilities. Someone executing a task on behalf of someone is legal but suspicious and deserves investigation. We introduce a predicate to capture this:

$$\begin{aligned}&\forall\ u : User; ta : Task; s : Session; t : Time \mid \\ &\quad HoldsAt(ExecutedTaskAsDelegator(u, ta, s), t) \\ &\quad \Leftrightarrow \exists\ u_2 : User; t_2 : Time \mid t_2\ \leq\ t \\ &\qquad Happens\ (ExecTask\ (ta, u_2, s), t_2) \\ &\qquad \wedge\ HoldsAt\ (HasDelegTo\ (u, u_2), t_2) \\ &\qquad \wedge\ HoldsAt(CanExecAsDelegateOnly(u_2, ta), t_2))\end{aligned}$$

From this predicate, the goal is defined by describing states of complete workflow runs where someone executes a task on behalf of someone else. This results in the goal:

$$\begin{aligned}&\exists\ u : User; ta, : Task; s : Session; t : Time \mid \\ &\quad HoldsAt\ (IsWrkfComplete\ (s), t) \\ &\quad \wedge HoldsAt(ExecutedTaskAsDelegator(u, ta, s), t)\end{aligned}$$

The plans generated by *decreasoner* result in what is normally expected under delegation (someone executes a task on behalf of someone else), but they also result in plans that may be possible frauds: a delegate approves a payment on behalf of the manager and the same manager also approves the same payment. From this, we elaborate the requirements, and we get $R6'$ (table 4) an elaboration of requirement $R6$. When the model is fixed, this possible fraudulent behaviour is no longer allowed.

7. Experimental Results

In both experiments presented above, formal analysis verified that it was not possible to reach an unsecure state in the modelled systems, where security requirements would be breached. However, suspicion-based analysis demonstrated that the modelled systems were in fact not secure.

Section 5 analyses a simple medical information system that includes a confidentiality requirement. Following the traditional route of safety analysis, it was not possible to find ways in which confidentiality would be compromised:

Case Study	Anaysis Goal	Time
SMIS	Security Violation	5.2s
SMIS	Suspicion goal 1	6.1s
SMIS	Suspicion goal 1, after fix	28.2s
Workflow	Security violation	314s (5.2m)
Workflow	Suspicion goal 1	692.3s (11.5m)
Workflow	Suspicion goal 1, after fix	618.6s (10.3m)
Workflow	Suspicion goal 2	353.3s (5.9m)
Workflow	Suspicion goal 2, after fix	360.8s (6.0m)

Table 5. Running times for analysis of EC models with *decreasoner*. Table indicates case study, analysis goal and time it takes to generate plans.

without a valid credential it would not be possible to obtain the patient's medical data. Analysis based on suspicion then uncovered a security vulnerability (or loophole) that would enable a malicious user to obtain the required credentials in a non-legal way.

Section 6 analyses a business process whose security requirements included two integrity requirements enforced through SoD. Again, SoD could be breached, but not in an obvious way. Following the traditional safety analysis route, we checked that it was not possible that the same user would be able to execute two different tasks protected by SoD. Analysis-based on suspicion, however, uncovered several problems: the same user could execute different tasks in a workflow session under different roles, and delegation introduced a *loophole* that would enable users to indirectly breach SoD.

Table 5 presents the running times of the formal analysis based on planning with *decreasoner*[2]. For each case study, it shows how much time it took to carry out the analysis for each analysis goal. We can see that the analysis of the workflow model of section 6 takes substantially longer than SMIS analysis because it is more complex.

8. Discussion

This paper proposes *suspicion* to drive the analysis of security requirements. Through experiments, it argues that in security the interesting question is not only to know whether there is a state compromising some security property (safety analysis), but also to look for what is suspicious in order to find security vulnerabilities and threats. The experiments are conducted in the context of the EC temporal logic, planning and the *decreasoner* tool. They demonstrate the usefulness of suspicion. The tradition safety analysis route, which checks whether some security property is violated, would not expose any security issues; this can mislead analysts in concluding that the system being analysed is secure. Analysis based on suspicion uncovered security vulnerabilities and threats; such findings drive elaboration of the requirements.

The approach presented here generates automatically possible scenarios of misuse (threats) from a statement describing some security violation or suspicious condition (the goal). This provides a flexible and illuminating scheme to the analysis of security requirements. Rather then finding themselves possible threats, analysts describe instead what would constitute a violation of security or a suspicious system condition. Analysis goals require an understanding of the requirements domain, and should be described with some security asset in mind.

The security vulnerabilities exposed by the analysis may seem contrived, but they illustrate the sort of vulnerabilities that attackers exploit to intrude into today's software systems. The vulnerabilities identified in the health-care system give insiders the opportunity to perpetrates attacks on the system; the *insider threat* has been identified as one of the main sources of attacks in the medical domain [16]. Once a source of threats is identified in the requirements, two decisions can be made: (a) introduce further constraints by elaborating the requirements so that the source of threats is eliminated, or (b) do nothing in terms of requirements, but take the problem into account in terms of run-time intrusion and threat detection which then has to judge whether some uses of the system are malicious or not. The latter must be considered because it is not possible to eliminate all possible security threats, doing so could result in a system design that is rigid and over-constrained.

The experiments conducted here confirm the importance of modelling and analysing security together with system requirements. Both case studies show how a functionality of the system, delegation, have a serious impact on security and how it was necessary to further elicit and elaborate the requirements in order to eliminate threats. It is interesting that many formal models tend to include some security mechanism (such as role-based access control) in the model of the requirements. Here, the requirements were modelled independently of any design decisions as much as possible, so that abstract security requirement models can be refined into particular security mechanism. Inevitably, some design decision have to be made at the level of the more abstract model; the aim was to keep this to a minimum.

Formal security analysis with tool support is capable of exposing many unexpected situations, providing a level of assurance not guaranteed by semi-formal approaches. The drawback of the analysis with *decreasoner* lies in the efficiency of the tool: as models get more complex, the solution to analysis problems take more time to the point that the analysis becomes unpractical. The workflow model of section 6 is a simplification of an earlier model to enable practical analysis. As usual, the secrete is in getting the right abstraction in order to analyse the property of interest.

It is interesting to comment on the usability of the

2. Model analysis carried out on an Apple *MacBook*, with a 2.2 Ghz Intel Core Duo processor and 2GB memory RAM.

approach presented here. The process of defining analysis goals may require domain knowledge and skill in building and analysing models. However, the process can be partially or fully automated by following the pattern-based approach proposed of [11]. [11] uses the *Formal Template Language* [19], [13] to represent patterns of EC models together with their associated security monitoring goals. [11] defines templates of such goals that are of the security violation type, but patterns of suspicious goals can also be defined if we know in advance what can arouse suspicion. In our experiments, *delegation* was the focus of our suspicious goals; this is something that can be known in advance and captured using patterns. Following [11], we can have goals that capture what is known to arouse suspicion; actual suspicious goals would then be automatically generated from templates. [11] also uses UML models to enable intuitive requirements modelling; the same approach can also be used to enhance the usability of the approach proposed here.

9. Conclusions

This paper proposes an approach to the formal analysis of security requirements based on planning guided by the concept of suspicion. The approach was illustrated using the EC and the *decreasoner tool* by performing two experiments: one involving a simple health-care system with a confidentiality requirement and another a business system with an integrity requirement enforced through SoD. The experiments showed how functional requirements can impact upon security requirements in subtle ways. It showed that the more obvious way of analysing security, by doing the traditional safety analysis would not give any useful results: an unsecure state could not be reached. However, it was through more flexible goals based on suspicion that we could obtain useful results exposing subtle security vulnerabilities.

The main contribution of this paper is the proposal of suspicion as a driving concept in the analysis of security requirements. The paper also provides experimental evidence to certain claims made in the security requirements literature. It confirmed that it is important to model security requirements together with its system counter-parts because functionality impacts on security. It showed that a system verified against some security property does not necessarily mean that it is secure; often, more flexible analysis driven by concerns such as what is suspicious is more useful because it exposes subtle security weaknesses. This paper also demonstrated the importance of formality and tool support, and usefulness of the AI technique of planning in the analysis of requirements.

References

[1] P. T. Devanbu and S. Stubblebine, "Software engineering for security: A roadmap," in *The Future of Software Engineering*. ACM, 2000, pp. 227–239.

[2] B. W. Boehm, "Software engineering," *IEEE Transactions on Computers*, pp. 1266–1241, 1976.

[3] A. van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," in *Proc. ICSE'04*, 2004, pp. 148–157.

[4] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE Trans. on Software Engineering*, vol. 3, no. 2, pp. 125–143, 1977.

[5] M. Shanahan, "The event calculus explained," in *Artificial Intelligence Today*, ser. LNCS. Springer, 1999, vol. 1600, pp. 409–430.

[6] L. Lin, B. Nuseibeh, D. Ince, and M. Jackson, "Using abuse frames to bound the scope of security problems," in *Proc. RE '04*. IEEE, 2004, pp. 354–355.

[7] I. Alexander, "Misuse cases: Use cases with hostile intent," *IEEE Software*, vol. 20, no. 1, pp. 58–66, 2003.

[8] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *Annual computer security applications conference*. IEEE, 1999.

[9] D. Denning, "An intrusion detection model," *IEEE Trans. on Software Engineering*, vol. 13, no. 2, pp. 222–232, 1987.

[10] T. Hollebeek and R. Waltzman, "The role of suspicion in model-based intrusion detection," in *Proc. of NSPW '04*. ACM, 2004, pp. 87–94.

[11] N. Amálio and G. Spanoudakis, "From monitoring templates to security monitoring and threat detection," in *Proc. of SECURWARE '08*. IEEE, 2008, pp. 185–192.

[12] N. Amálio, S. Stepney, and F. Polack, "Formal proof from UML models," in *Proc. ICFEM 2004*, ser. LNCS, vol. 3308. Springer, 2004, pp. 418–433.

[13] N. Amálio, "Generative frameworks for rigorous model-driven development," Ph.D. dissertation, Dept. Computer Science, Univ. of York, 2007.

[14] J. Allen, J. Hendler, and A. Tate, Eds., *Readings in planning*. Morgan Kaufmann, 1990.

[15] E. T. Muller, "Event calculus reasoning through satisfiability," *Journal of Logic and Computation*, vol. 14, no. 5, pp. 703–730, 2004.

[16] R. J. Anderson, "A security policy model for clinical information systems," in *Proc. of SP '96*. IEEE, 1996.

[17] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *Proc. IEEE Symp. Research in Security and Privacy*, 1987, pp. 184–194.

[18] M. J. Nash and K. R. Poland, "Some conundrums concerning separation of duty," in *Proc. IEEE Symp. Research in Security and Privacy*, 1990, pp. 201–207.

[19] N. Amálio, S. Stepney, and F. Polack, "A formal template language enabling meta-proof," in *FM 2006*, ser. LNCS, vol. 4085. Springer, 2006, pp. 252–267.