2nd International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2008)

From monitoring templates to security monitoring and threat detection

Nuno Amálio and George Spanoudakis
Dept. Computing, City University London
London, UK
{nuno.amalio, gespan}@soi.city.ac.uk

Abstract. *This paper presents our pattern-based approach to run-time requirements monitoring and threat detection being developed as part of an approach to build frameworks supporting the construction of secure and dependable systems for ambient intelligence. Our patterns infra-structure is based on templates. From templates we generate Event-Calculus formulas expressing security requirements to monitor at run-time. From these theories we generate attack signatures, describing threats or possible attacks to the system. At run-time, we evaluate the likelihood of threats from run-time observations using a probabilistic model based on Bayesian networks.*

Keywords

Security, patterns, intrusion-detection, run-time monitoring, Event-Calculus.

1. Introduction

The vision of Ambient Intelligence (AmI) [1] poses many technological challenges. The European research project SERENITY[1] tries to contribute to the realisation of this vision by developing an approach to construct AmI ecosystems that are both secure and dependable. The SERENITY effort proposes an architecture to construct design and run-time frameworks for AmI [2].

In SERENITY, patterns have a pivotal role. They factor expertise and can be used in a variety of contexts. Our role in SERENITY is run-time systems monitoring. In particular, we are concerned with checking that security and dependability (S&D) solutions emerging from SERENITY frameworks satisfy their S&D requirements at run-time.

Like the whole run-time SERENITY framework, its run-time monitoring component needs to adapt to a variety of contexts following a pattern-based approach. For this purpose, we use a very concrete form of pattern: templates. Our run-time monitoring component comprises a catalogue of security monitoring specification templates that are instantiated to provide S&D monitoring solutions in a variety of contexts. The aim of this template infra-structure is two-fold: (a) factor and reuse expertise in building monitoring specifications; (b) enable automation so that monitoring specifications are automatically generated from other artefacts of the SERENITY framework.

Due to their distributed nature, AmI systems are vulnerable to attacks: the attacker can be anywhere and choose the easiest entry [3]. One of the aims of the SERENITY run-time framework, of which our monitoring component is part of, is to be able to detect attacks and predict threats (the possible occurrence of attacks).

Previously, we developed a system to monitor requirements of service-based systems [4] where requirements are expressed in Event Calculus. We have also advocated an approach to security monitoring based on patterns in the context of SERENITY [5, 6].

This paper takes this work further and presents part of our concrete approach to S&D run-time monitoring in SERENITY. In particular, it presents:

- The approach driving the generation of EC formulas used for requirements monitoring and threat detection, which includes the mechanism underlying the transition from design and architectural models to a specific monitoring theory.
- The approach to detect threats to security and dependability requirements at run-time.

The following starts by giving some background on the languages Event Calculus and FTL. Section 3 presents the example that is used to motivate and illustrate our approach. Section 4 gives a brief overview of our overall approach to security monitoring and threat detection from templates. Section 5 presents shows how monitoring specifications are generated from monitoring templates. Section 6 shows how attack signatures are generated from a monitoring specification. Section 7 shows our approach to calculate the probability of threats at *run-time*. The remaining sections discuss the results presented in this paper, compare our approach with related work and take the conclusions of the paper.

2. Background

Our approach needs a way to express security monitoring specifications and templates of these specifications. This is done using the Event Calculus and the Formal Template Language (FTL).

2.1 The Event Calculus

The Event Calculus (EC) [7] is a language based on first-order predicate-calculus that is designed to represent and reason about action and change. The basic ontology of EC comprises *events*, *fluents* and *timepoints*. An *event* represents an action that may occur in the world. A *fluent* represents a time varying property of the world. A *timepoint* represents an instance of time. C includes a set of basic predicates to describe happening of events, their effects and the state of fluents.

[1] http://www.serenity-project.org

The basic predicates of EC are as follows:

- *HoldsAt (f, t)* says that fluent *f* is true at time-point t.
- *Happens (e, t, R(t_1, t_2))* says that event *e* may occur at time-point t within time range $t_1 \ldots t_2$.
- *Initiates (e, f, t)* says that if event *e* occurs at time point *t*, then fluent *f* is true after *t*.
- *Terminates (e, f, t)* says that if event *e* occurs at time point *t*, then fluent *f* is false after *t*.
- *Initially$_P$ (f)* says that fluent f holds at timepoint 0.

2.2 FTL

To express templates we use the Formal Template Language (FTL) [8, 9], a generic formal language for expressing templates of any target language. A key characteristic of FTL is that it is *generative*; FTL describes sentences of some target language (here we use EC) and can generate sentences when provided with an instantiation. The main constructs of FTL are *placeholders*, *lists* and *choice*.

To illustrate FTL, suppose the following FTL template of an EC predicate, which specifies a number of event preconditions associated with the template event *E*:

$$(\forall t : Time)\ Happens(\langle E\rangle, t, R(t, t)) \Rightarrow \mathbf{[}HoldsAt(\langle F\rangle, t)\mathbf{]}$$

This template includes two placeholders and one list term. It basically says that a number of pre-conditions (*HoldsAt* predicate inside the list with placeholder *F*) may be associated with some event (*E* placeholder in *Happens* predicate). The template can be instantiated to give:

$$(\forall t : Time)\ Happens(Eat, t, R(t, t)) \Rightarrow$$
$$HoldsAt(IsHungry, t) \wedge HoldsAt(DinnerServed, t)$$

3. Motivating Example

To illustrate the approach presented here, we use a case study of an e-health care system used in the SERENITY project [10]. Our simple e-health care system enables doctors to access the medical data of patients by digital means; its requirements are summarised in Table 1. Essentially, this system provides an operation to access the full contents of a patient's medical file (*R1*), and an operation to access the partial contents of a patient's medical file (*R2*).

R 1	A patient's medical file may be seen by the doctor of the patient only.
R 2	All doctors may see partial pieces of information belonging to a patient's medical file in a way that preserves requirement R1.

Table 1: The requirements of the e-health care system (based on [10]).

Requirement *R1* is a confidentiality security requirement protecting the privacy of patients. The functionality introduced by requirement *R2*, however, may conflict with *R1*. *R2* says that there are indirect means to access the information of a patient's medical file, but this must be done with care in order not to breach *R1*.

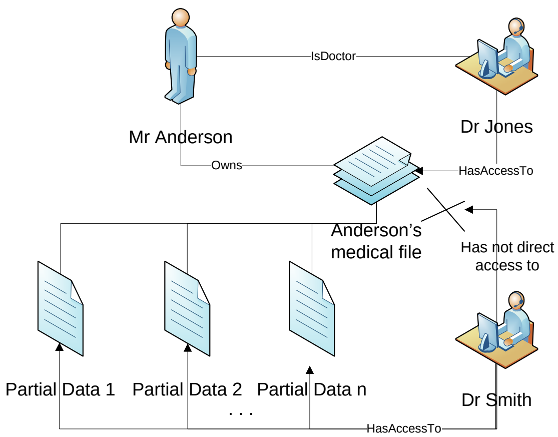

Figure 1: A scenario of the e-health care system, where Dr Smiths infers the medical file of someone else's patient.

Figure 1 depicts two scenarios illustrating the simple e-health care system and its security requirements. We have doctor Jones, his patient Mr Anderson, and another doctor, Smith, who is not Anderson's doctor. According to R1, Dr Jones has direct access to Anderson's medical file, but Smith does not because he is not Anderson's doctor. Figure 1 also illustrates how *R2* may impact on *R1*. We can see that Dr Smith, although not Anderson's Doctor, is able to infer Anderson's medical file by indirect means breaching requirement *R1*. This indirect way of accessing confidential data is an instance of the inference attacks [11].

4. Overview of the approach

Security requirements aim to protect some system from malicious attacks. Our approach checks that such requirements are preserved at run-time by monitoring the satisfaction of EC formulas that formalise them. This is done by observing the system at run-time and checking observations against specified system behaviour trying to detect deviations from what is specified. There are two different types of EC formulas: *monitoring rules* and *assumptions*. Monitoring rules are those formulas whose satisfaction is checked at run-time. Assumptions specify rules on how to derive information about the state of the system that is being monitored based on observations of its behaviour. A collection of these EC formulas is called *monitoring specification*.

A monitoring specification effectively describes the security policy of the system. When an attack takes place, we consider that the security policy is violated and, therefore, a monitoring rule is breached. This means that our requirements monitoring approach is capable of detecting immediate attacks to the system, or, what is referred in the security literature as *intrusions* [12-14]: some monitoring rule is breached and our tool detects this. A threat, however, is a possible attack to the system [13]; to detect them our approach needs to be *one-step-ahead* and foresee future violations of monitoring rules.

Attackers are constantly looking for new ways of attacking systems. We somehow need to devise a strategy to prevent this. Our strategy is based on the security concept of an *asset*. Assets are the resources that a system must protect from incorrect or unauthorised use [15]. They are attack targets; the motivation of an attack to the system. Our security templates are designed to protect some system asset.

The following discusses the process involved in the generation of monitoring specifications and attack signatures, and overviews our approach to threat detection.

4.1 The Generation process

In SERENITY, every EC formula used for threat detection is generated from a security template expressed in FTL, which expresses a basic security property, namely, *confidentiality*, *integrity* and *availability*. As violations of these properties would breach system security, we use the templates to derive possible attack signatures over assets.

The whole process of generation of a monitoring specification is triggered by the pair ⟨Security Objective, Asset⟩. The whole process, depicted in Figure 2, is as follows:

1. The template is selected from the security objective (refers to template name).
2. The system asset is used to select a set of SERENITY artefacts that refer to it.
3. Information derived from selected artefacts are used to instantiate the selected monitoring template, resulting in the generation of monitoring specification.
4. The attack signature is generated from the monitoring specification.
5. Finally, the threat probabilistic model is generated from the attack signature.

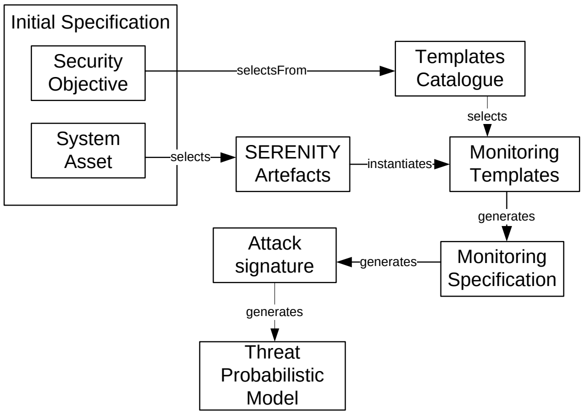

Figure 2: Generation of Monitoring specification, attack signature and threat probabilistic model.

4.2 A threat is a possible violation of a monitoring rule

Our approach to threat detection starts from an EC monitoring specification. Since a threat is a possible violation of a security monitoring rule, we try to predict this in advance by using AI planning techniques [16, 17]. We derive an abstract plan describing all possible sequences of events in which the monitoring rule is violated. We call this abstract plan an *attack signature*, a formula describing all possible attacks that lead to the violation of the security monitoring rule.

An actual attack is a sequence of events that have been observed, something that happened. A threat is a possible attack, a sequence of events that has not yet happened but that may happen. In our approach, we compute the attack signature at startup (before monitoring starts). At run-time, all observed sequences of events that match the signature constitutes either a threat (the sequence partially matches the sequence) or an attack (the sequence of events totally matches the sequence).

It is uncertain whether a threat is going to materialise itself as an attack or not. To evaluate how likely it is that a threat is going to materialise itself as an attack we resort to probability theory. We use a probabilistic model based on the Bayesian theory of probability [18]. Initially, when some event that matches the attack signature is observed then this constitutes a possible threat and its likelihood is evaluated. As more observations are gathered, the likelihood of threats may be confirmed and become stronger or refuted and become weaker; these evaluations are performed using the Bayesian rule of conditional probability.

5. Generation of monitoring specifications

This section presents a scheme for the generation of monitoring specifications by instantiating templates. The following introduces a confidentiality monitoring template, and illustrates the generation process from this template using a design model of our case study.

5.1 Confidentiality Monitoring Template

The core confidentiality monitoring template, given in Figure 3, comprises one monitoring rule (formula 1) and assumptions (formulas 2–3). Formula 1 says that an agent may be exposed to some asset provided he is authorised to do so. Formula 2 says that the template event "*⟨Authorise⟩*" initiates (sets to true) template fluent "*Authorised⟨Asset⟩*". Formula 3 says that the template event "*⟨Disclose⟩*" initiates template fluent "*Exposed⟨Asset⟩*".

The template for confidentiality with inference, which extends the core template to allow indirect access to the asset, is given in Figure 4. This template is made entirely of template assumptions. In the template, formula 4 says that the template event "*⟨DiscloseP⟩*" initiates the template fluent "*KnowsPD⟨Asset⟩*". Formula 5 says that event "*⟨DiscloseP⟩*" initiates the template fluent "*Exposed⟨Asset⟩*" provided the agent already knows all partial pieces of the medical file except the one being accessed (by event *⟨DiscloseP⟩*). Finally, template formula 6 specifies which actual partial pieces of data are available for some asset (template fluent *availablePD⟨Asset⟩*).

$$\langle Confidentiality \rangle_{def} ==$$

$$(\forall ag : \langle Agent \rangle;\ a : \langle Asset \rangle;\ t : Time) \qquad (1)$$
$$HoldsAt\ (Exposed\langle Asset \rangle\ (ag, a), t)$$
$$\Rightarrow HoldsAt\ (Authorised\langle Asset \rangle (ag, a), t)$$

$$(\forall ag : \langle Agent \rangle;\ a : \langle Asset \rangle;\ t : Time) \qquad (2)$$
$$Happens(\langle Authorise \rangle\ (ag, a), t, R\ (t, t))$$
$$\Rightarrow\ Initiates(\langle Authorise \rangle\ (ag, a),$$

$$\begin{array}{ll}
\qquad \mathit{Authorised}\langle \mathit{Asset}\rangle\ (ag, a), t) & \\
(\forall ag : \langle \mathit{Agent}\rangle;\ a{:}\ \langle \mathit{Asset}\rangle;\ \ t : \mathit{Time}) & (3)\\
\quad \mathit{Happens}(\langle \mathit{Disclose}\rangle\ (ag, a), t, R\ (t, t)) & \\
\qquad \Rightarrow \mathit{Initiates}\ (\langle \mathit{Disclose}\rangle\ (a, ag), & \\
\qquad\ \ \mathit{Exposed}\langle \mathit{Asset}\rangle\ (ag, a), t) &
\end{array}$$

Figure 3: Core confidentiality template.

$$\begin{array}{ll}
\langle \mathit{Confidentiality.Inference}\rangle_{def} == & \\
\langle \mathit{Confidentiality}\rangle_{ref} & \\
\forall ag : \langle \mathit{Agent}\rangle;\ a{:}\ \langle \mathit{Asset}\rangle;\ pdt : \langle \mathit{PDT}\rangle;\ t : \mathit{Time}) & (4)\\
\quad \mathit{Happens}(\langle \mathit{DiscloseP}\rangle\ (ag, a, pdt), t, R\ (t, t)) & \\
\quad \wedge\ \mathit{HoldsAt}(\mathit{AvailablePD}\langle \mathit{Asset}\rangle\ (pdt, t) & \\
\qquad \Rightarrow\ \mathit{Initiates}\ (\langle \mathit{DiscloseP}\rangle\ \ (ag, a, pdt), & \\
\qquad\quad \mathit{KnowsPD}\langle \mathit{Asset}\rangle\ (ag, a, pdt), t) & \\
(\forall ag : \langle \mathit{Agent}\rangle;\ a{:}\ \langle \mathit{Asset}\rangle;\ pdt : \langle \mathit{PDT}\rangle;\ t : \mathit{Time}) & (5)\\
\quad \mathit{Happens}\ (\langle \mathit{DiscloseP}\rangle\ (ag, a, pdt), t, R\ (t, t)) & \\
\quad \wedge\ \mathit{HoldsAt}(\mathit{AvailablePD}\langle \mathit{Asset}\rangle\ \ (pdt), t) & \\
\quad \wedge\ ((\forall pdt2 : \langle \mathit{PDT}\rangle)\ pdt \neq pdt2 & \\
\quad \wedge\ \mathit{HoldsAt}(\mathit{AvailablePD}\langle \mathit{Asset}\rangle\ (pdt2), t) & \\
\qquad \Rightarrow \mathit{HoldsAt}(\mathit{KnowsPD}\langle \mathit{Asset}\rangle\ (ag, a, pdt2), t)) & \\
\quad \Rightarrow\ \mathit{Initiates}(\langle \mathit{DiscloseP}\rangle\ (ag, a, pdt), & \\
\qquad \mathit{Exposed}\langle \mathit{Asset}\rangle\ (a, ag), t) & \\
\mathbf{[}\mathit{Initially}_P(\mathit{AvailablePD}\langle \mathit{Asset}\rangle(\langle \mathit{PD}\rangle))\mathbf{]} & (6)
\end{array}$$

Figure 4: Extension to core template for monitoring access to partial data.

5.2 Generation from security objective and template instantiation

We now show how the templates of Figure 3 and Figure 4 are instantiated to produce a monitoring specification. We start from the object/asset pair:

$$\langle \mathit{Confidentiality.Inference},\ \mathit{Patient}\rangle$$

This says that we want to monitor the “Confidentiality.Inference” security property over the “Patient” asset. The first element of the pair (security property) results in the selection of the template of Figure 4. The second element of the pair (*Patient* asset) would result in the selection of a set of design models that specify properties of the asset. For this effect, we introduce the simple design UML class model of Figure 5.

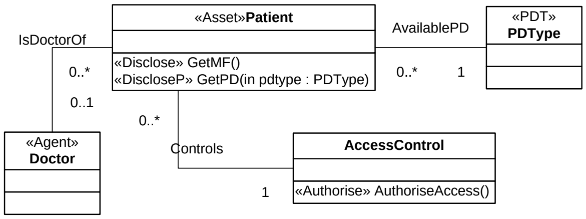

Figure 5: A simple system for doctors and their patients.

The class model of Figure 5 introduces the classes *Doctor*, *Patient*, *PDType* and *AccessControl*. This is used to instantiate the template; the link is established through stereotypes: each stereotype corresponds to the name of a placeholder.

The class model above enables the automatic generation of the EC theory of Figure 6, where the template is instantiated with the substitution set:

$$\begin{array}{c}
\{\mathit{Agent} \mapsto \text{``}\mathit{Doctor}\text{''},\ \mathit{Asset} \mapsto \text{``}\mathit{Patient}\text{''},\ \mathit{Disclose} \mapsto \text{``}\mathit{GetMF}\text{''},\\
\mathit{Authorise} \mapsto \text{``}\mathit{AuthoriseAccess}\text{''},\\
\mathit{PDT} \mapsto \text{``}\mathit{PDType}\text{''},\ \mathit{DiscloseP} \mapsto \text{``}\mathit{GetPD}\text{''},\\
\mathit{PD} \mapsto \{\text{“}\mathit{PD1}\text{”},\ \text{“}\mathit{PD2}\text{”},\ \text{“}\mathit{PD3}\text{”}\}\}
\end{array}$$

As it can be seen, this substitution set is derived from the information contained in the diagram.

$$\begin{array}{ll}
(\forall ag : \mathit{Doctor};\ a{:}\ \mathit{Patient};\ \ t : \mathit{Time}) & (7)\\
\quad \mathit{HoldsAt}\ (\mathit{ExposedPatient}\ (ag, a), t) & \\
\qquad \Rightarrow \mathit{HoldsAt}\ (\mathit{AuthorisedPatient}(ag, a), t) & \\
(\forall ag : \mathit{Doctor};\ a{:}\ \mathit{Patient};\ \ t : \mathit{Time}) & (8)\\
\quad \mathit{Happens}(\mathit{AuthoriseAccess}(ag, a), t, R\ (t, t)) & \\
\qquad \Rightarrow \mathit{Initiates}\ (\mathit{AuthoriseAccess}(ag, a), & \\
\qquad\quad \mathit{AuthorisedPatient}\ \ (ag, a), t) & \\
(\forall ag : \mathit{Doctor};\ a{:}\ \mathit{Patient};\ \ t : \mathit{Time}) & (9)\\
\quad \mathit{Happens}(\mathit{GetMF}\ (ag, a), t, R\ (t, t)) & \\
\qquad \Rightarrow \mathit{Initiates}\ (\mathit{GetMF}\ (a, ag), \mathit{ExposedPatient}\ (ag, a), t) & \\
\forall ag : \mathit{Doctor};\ a{:}\ \mathit{Patient};\ pdt : \mathit{PDType};\ t : \mathit{Time}) & (10)\\
\quad \mathit{Happens}\ (\mathit{GetPD}\ (ag, a, pdt), t, R\ (t, t)) & \\
\quad \wedge\ \mathit{HoldsAt}(\mathit{AvailablePDPatient}\ (pdt), t) & \\
\qquad \Rightarrow \mathit{Initiates}\ (\mathit{GetPD}\ (ag, a, pdt), & \\
\qquad\quad \mathit{KnowsPDPatient}\ (ag, a, pdt), t) & \\
(\forall ag : \mathit{Doctor};\ a{:}\ \mathit{Patient};\ pdt : \mathit{PDType};\ t : \mathit{Time}) & (11)\\
\quad \mathit{Happens}(\mathit{GetPD}(ag, a, pdt), t, R\ (t, t)) & \\
\qquad \wedge\ \mathit{HoldsAt}(\mathit{AvailablePDPatient}(pdt), t) & \\
\qquad \wedge\ ((\forall pdt2 : \mathit{PDType})\ pdt \neq pdt2 & \\
\qquad \wedge\ \mathit{HoldsAt}(\mathit{AvailablePDPatient}\ (pdt2), t) & \\
\qquad\quad \Rightarrow \mathit{HoldsAt}(\mathit{KnowsPDPatient}\ (ag, a, pdt2), t)) & \\
\qquad \Rightarrow \mathit{Initiates}(\mathit{GetPD}(ag, a, pdt), \mathit{ExposedPatient}\ (a, ag), t) & \\
\mathit{Initially}_P(\mathit{AvailablePDPatient}(PD1)) & (12)\\
\mathit{Initially}_P(\mathit{AvailablePDPatient}(PD2)) & (13)\\
\mathit{Initially}_P(\mathit{AvailablePDPatient}(PD3)) & (14)
\end{array}$$

Figure 6: Event Calculus monitoring specification generated by instantiating monitoring template with information coming from the class diagram of Figure 5.

6. Generation of attack signatures

Attack signatures are generated from a monitoring theory by using AI planning techniques [16, 17]. Classical planning derives a concrete sequence of actions to achieve a *goal*. Our task does, however, differ slightly from the work on classical planning. We need both *conditional* and *partial-order* plans, not just a sequence of actions. Conditional because not all the information is available when we derive the plans and we may have to consider different branches [17]. Partial-order because we want to obtain plans with a partial-order on actions [16].

We intend to use an abductive strategy towards planning [17]. In particular, we intend to adapt the algorithm of [19] that was designed for diagnostics to planning.

Our proposed approach is as follows:

- We negate the security monitoring rule of the EC monitoring specification (there is only one).
- We derive a plan signature given the monitoring specification, and the goal.

Given the EC theory of Figure 6, we start by deriving the goal, which is obtained by negating equation 7:

$(\exists\, ag : Doctor;\ a\!: Patient;\ \ t : Time)$
$\quad HoldsAt\ (ExposedPatient\ (ag, a), t)$
$\quad \wedge \neg HoldsAt\ (AuthorisedPatient(ag, a), t)$

This says that the goal is a state where the patient's medical file is exposed and the doctor accessing it is not authorised.

An attack signature is a conditional or partial order plan describing how the goal may be achieved. Given the goal above, we follow an abductive planning approach (backwards from goal) to generate the signature:

$(\exists\, ag : Doctor;\ a : patient,\ t1, t2, t3, t4, t5, t6 : Time)$
$\quad (Happens(GetMF(ag, a), t1)$
$\quad \wedge\neg Happens(AuthoriseAccess(ag, a), t2) \wedge t2 < t1)$
$\quad \vee (Happens(GetPD(ag, a,PD1), t3)$
$\qquad \wedge\ Happens(GetPD(ag, a, PD2), t4)$
$\qquad \wedge\ Happens(GetPD(ag, a, PD3), t5)$
$\qquad \wedge\neg Happens(AuthoriseAccess(ag, a), t6)$
$\qquad \wedge\ t6 < t3 \wedge t6 < t4 \wedge t6 < t5)$

This EC formula is a disjunction, where each disjunct is one possible attack. This formula identifies two attacks:

- Either an unauthorised user accesses the medical file directly through the operation *GetMF*,
- or a user accesses the medical file indirectly through the operation *GetPD* (Inference).

7. Calculating the probability of threats

Our probabilistic model for threat evaluation is set in the framework of Bayesian Networks (BNs) [18]. We assemble a generic BN from the attack signature, and use this network at run-time to evaluate the likelihood of threats. When some event is observed the probability that an attack is likely to materialise is updated using Bayes's rule of conditioning.

The following explains how we can assemble a BN for the attack signature derived above, and illustrates the network in calculating probabilities of threats from runtime observations.

7.1 Assembling the Bayesian network

Figure 7 presents the generic BN assembled from the attack signature above. The node *Attack* in the BN measures how likely it is that an attack will materialise. Our attack signature (see above), identifies two possible attacks: d*irect access* and *inference*. These two attacks are represented with a node in the BN, and they are connected to the *Attack* node: an *Attack* is either caused by *DirectAccess* or by *Inference*. An instance of these attacks occurs when we observe the events identified in the signature. These are represented in the BN with a causal arrow from the specific attack to the events that make it happen. For instance, there are two events that drive the occurrence of the *direct access* attack: *GetMF* and *AuthoriseAccess*. Note that *AuthoriseAccess* is a *negative* event; an instance of *DirectAccess* or *Inference* happens provided *AuthoriseAccess* is not observed. This is reflected in the prior probabilities (see below) [2].

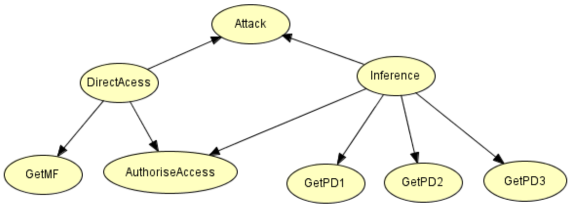

Figure 7. The Bayesian network for the two possible attacks: direct access and inference.

We assume that the *DirectAccess* (DA) and *Inference* (I) attacks are equally likely: *P(DA) = P(I) = 0.1*. The prior probability of Attack (A) conditioned on both *DA* and *I* — P (A| DA, I) — is given in Table 2. If either *DA* or *I* are true then *A* is also true with probability 1; if both are false, then A is true with probability 0 (an attack does not take place). Table 3 gives the probability of event *AuthoriseAccess* (*AU*) given *DA* and *I* — P(AU|DA, I). If either *DA* or *I* have materialised (they are *true)* then it is certain that *AU* must not have happened (*false* has probability 1). If they have not materialised, then there is a probability of *0.1* that *AU* has happened. Table 4 presents the prior probability of the event G*etMF* given DA. If *DA* has materialised then *GetMF* must have been observed — P(GetMF = true | DA = true) = 1 —, otherwise there is a marginal probability of 0.01 that *GetMF* has been observed. The prior probabilities of *GetPD1*, *GetPD2*, and *GetPD3* conditioned on I are as Table 5.

	DA	*true*		*false*	
	I	*true*	*false*	*true*	*false*
A	*true*	1	1	1	0
	false	0	0	0	1

Table 2. The probabilities of an attack (A) conditioned on *DA* and *I*: *P(A | DA, I)*.

	DA	*true*		*false*	
	I	*true*	*false*	*true*	*false*
AU	*true*	0	0	0	0.1
	false	1	1	1	0.9

Table 3. The probability of the event *AuthoriseAccess* (AU) conditioned on *DA* and *I*: *P (AU | DA, I)*.

	DA	*true*	*false*
GetMF	*true*	1	0.01
	false	0	0.99

Table 4. The probability of the event *GetMF* conditioned on *DA*: *P(GetMF | DA)*.

	I	*true*	*false*
GetPDi	*true*	1	0.1

[2] We use the term *GetMF* to denote the observation *Happens (GetMF(d, p), t, R(t, t))* for some doctor *d*, patient *p* and time point *t*. Likewise for the other events.

	false	0	0.9

Table 5. The probability of events *GetPDi* conditioned on *I*: *P(GetPDi | I)*.

7.2 Calculating threat probabilities at run-time

Given the tables and network above, we calculate the probabilities of an *Attack* and each specific attack (direct access or inference) from run-time observations.

In the scenario of Figure 1, suppose that Jones is in the system. At this point, an attack coming from Jones has a probability *0.19* (P(A)=0.19, P(DA) = P(I) = 0.1). But if Jones requests authorisation to access Anderson's medical file:

Happens (Authorise (Jones, Anderson), 1, R (1, 1))

The probability of an attack from Jones now drops dramatically (P(A) = P(DA) = P(I)=*0*). Now suppose that we observe Smith's activity in the system:

Happens (GetPD (Smith, Anderson, pdt1), 2, R (2, 2))

This results in an increase in the probability of an attack coming from Smith — P(A) = 0.5737 and P(I) = 0.5263. If Smith obtains another piece of data:

Happens (GetPD (Smith, Anderson, pdt2), 3, R (3, 3))

Then the probability of an attack from Smith increases even further — P(A) = *0.9174*, P(I) = *0.9257*. If Smith gets the third piece of data: P(A) = *0.9920*, P(I) = *0.9911*.

8. Discussion

Section 5 showed how EC formulas can be generated by using a simple UML class model. Our approach is also applicable to other kinds of descriptions. We have suggested the use stereotypes to link template placeholders to user models, but other schemes are possible. Provided the monitoring component receives an appropriate substitution set for some template, it is capable of generating EC formulas for monitoring. In our simple example, it would be possible to generate EC formulas automatically from a stereotyped UML diagram. This, however, is not always possible to achieve; and sometimes more sophisticated and complex means are required to enable an automatic transition from design models to run-time monitoring.

The concept of a security asset gives some flexibility to our approach to threat detection. Our attack signature does not come from some library of known attacks; instead it is generated by predicting what attackers will try to do from a specification of the system to monitor (the monitoring specification). Monitoring specifications come from security templates designed to protect generic system assets.

As illustrated in section 7, our probabilistic model handles negative evidence. The probability of an attack dropped dramatically when the event *AuthoriseAccess* was observed. The model also accounts for the cumulative effect of a sequence of observations that constitute positive evidence. In our example, the probability of an attack increases as more pieces of data are gathered.

Some of the prior probabilities of our probabilistic model may seem arbitrary, in particular those probability values that are neither 0 nor 1. We intend to improve this by using techniques to estimate conditional probabilities [20]; in particular techniques that learn conditional probabilities from new events [20].

We are trying to make our probabilistic model more accurate by considering further evidence. The idea is to associate *extra criteria* with our monitoring templates for the purpose of assessing the likelihood of threats. This would be just another collection of EC predicates that are monitored at run-time and that constitute evidence in favour or against the hypothesised threats.

9. Related Work

Our FTL-based approach brings our previous work on security monitoring patterns [6] into a fully formal setting. [6] proposes templates for confidentiality, availability and integrity. Here, we show how confidentiality templates can be described in FTL.

FTL is a formal language designed to represent and generate formulas of some target formal language, bringing pattern representations and their instantiation into a fully formal setting. This clearly contrasts to other approaches to security based on patterns, such as [21], which do not use formal representations of patterns. Other more formal approaches to notations for patterns and templates (see [8]) have less constructs than FTL and they are not generative.

Our approach and motivating example shares many similarities with the OrBAC approach to access control [22, 23]. OrBAC proposes a dynamic and flexible approach to security policy specification based on the concept of a *context* that is motivated by the health-care domain. It proposes a language based on first-order logic to express such security policies. Our security policies are expressed in the EC, a temporal logic based on first-order logic. Many ideas of OrBAC, including the notion of context, are expressible in EC. In addition, EC gives a temporal dimension that is explored in ORBAC, but lacks the level of sophistication that EC gives. Like OrBAC, our approach also follows the principle of adaption to a context, we introduce an extra level to allow this: templates. We adapt specifications to a context by instantiating templates.

Our approach to threat detection is related to *intrusion detection* [12-14]. Most intrusion detection systems, however, only detect malicious actions that have already happened (intrusions), which is what our requirements monitoring approach does (by detecting deviations from security requirements). Our approach to threat detection tries to be one-step ahead and predict malicious actions.

Approaches to intrusion detection are classified as *anomaly-based* or *misuse-based* [12]. Anomaly-based approaches [14, 24, 25] assume that attacks involve, somehow, abnormal behaviour of the system, and threats and intrusions are detected as deviations from normality. Misuse-based approaches [26-28], on the other hand, are based on models of known attacks. The threat detection approach presented here is essentially anomaly-based. In particular, it is *model* or *specification*-based [24, 25]: threats and intrusions are detected as deviations from a model of the normal behaviour of the system; it shares with [25] the concern of protecting system assets and in building security policies with the goal of protecting them. Our approach also has, however, characteristics of misuse-based techniques (we detect attacks from an attack signature, which effectively constitutes an attack model) and in particular statistical approaches based on Bayesian networks such as [28].

Our approach shares many similarities with [29]. In particular, the use of security specification patterns selected from some security goal and which are instantiated with information coming from object models, and the derivation of attack representations (called attack trees in [29]) from security specifications. The most striking difference, however, is that [29] is a design approach; feedback coming from attack analysis is fed back into the design. Our approach detects threats to S&D requirements at *run-time*. Other differences include the formalisms being used, and the fact that we use a probabilistic model to deal with uncertainty.

10. Conclusions

We have presented part of our approach to security monitoring in the context of the SERENITY approach. In particular, we showed how we can represent templates using the formal language FTL, and how this representation is amenable to automation: given an instantiation set we are able to generate an EC monitoring theory. We also presented our approach to threat detection where we use planning to compute an attack signature *off-line* (before monitoring starts), and how we use this signature to evaluate the likelihood of threats at run-time using a probabilistic model based on Bayesian networks.

Our approach was illustrated with a simple healthcare system. We showed how we could generate a monitoring theory from a confidentiality template and a design model. We then used this modelling theory to compute an attack signature that would violate the monitoring rule. Finally, we showed how we could evaluate the probability of threats given run-time observations by building a Bayesian network from the attack signature.

ACKNOWLEDGEMENTS

This work has been funded by the European commission as part of the project SERENITY (IST-027587). Christos Kloukinas provided useful feedback on this work.

REFERENCES

[1] Weiser, M., *The Computer for the 21st Century.* Scientific American, **265**(3). 1991.

[2] Sánchez-Cid, F., et al. *Software Engineering Techniques Applied to AmI: Security Patterns*. In *Developing Ambient Intelligence*. 108--123. Springer. 2006

[3] Verbauwhede, I., et al., *Security for Ambient Intelligent Systems*. In *Ambient Intelligence*, W. Weber, J.M. Rabaey, and E. Aarts, Editors. 2005, Springer. p. 199-221.

[4] Spanoudakis, G. and K. Mahbub, *Non Intrusive monitoring of service based systems.* Journal of Cooperative Information Systems, **15**(3): 325-358. 2006.

[5] Kloukinas, C. and G. Spanoudakis. *A pattern-driven framework for Monitoring Security and Dependability*. In *TrustBus'07*. 210--218. Springer. 2007

[6] Spanoudakis, G., C. Kloukinas, and K. Androutsopoulos. *Towards security monitoring patterns*. In *SAC '07: ACM symposium on Applied computing*. 1518--1525. ACM. 2007

[7] Shanahan, M. *The Event Calculus Explained*. In *Artificial Intelligence Today: Recent Trends and Developments*. Springer. Lecture Notes in Computer Science. 1999

[8] Amálio, N., S. Stepney, and F. Polack. *A formal template language enabling meta-proof.* In *FM 2006*. 252-267. LNCS, Springer. LNCS. 2006

[9] Amálio, N., *Generative frameworks for rigorous model-driven development*. PhD Thesis. Dept Computer Science, Univ of York. 2007.

[10] Campadello, S., et al. S&D Requirements specification. SERENITY PROJECT, SERENITY Deliverable, A7.D2.1

[11] Denning, D.E. and P.J. Denning, *Data Security.* ACM Comput. Surv., **11**(3): 227-249. 1979.

[12] Lazarevic, A., V. Kumar, and J. Srivastava, *Intrusion detection: a survey*. In *Managing cyber-threats: issues approaches and challenges.*, V. Kumar, J. Srivastava, and A. Lazarevic, Editors. 2005, Springer.

[13] Anderson, J.P. Computer Security Threat Monitoring and Surveillance. James P. Anderson Co., Technical Report,

[14] Denning, D., *An Intrusion Detection Model.* IEEE Transations on Software Engineering, **13**(2): 222-232. 1987.

[15] F. Swiderski and W. Snyder, *Threat Modeling*. 2004: Microsoft Press.

[16] Eshghi, K. *Abductive planning with Event Calculus*. In *5th International Conference on Logic Programming*. 562--579. MIT Press. 1988

[17] Levesque, H.J. *What is planning in the presence of sensing*. In *National Conference on Artificial intelligence (AAAI'96)*. 1139--1146. 1996

[18] Pearl, J., *Probabilistic reasoning in intelligent systems : networks of plausible inference*. 1988: Morgan Kaufmann.

[19] Console, P., L.a. Terenziani, and D.T. Dupre, *Local reasoning and knowledge compilation for efficient temporal abduction.* IEEE Transactions on Knowledge and Data Engineering, **14**(6): 1230 -1248. 2002.

[20] Niculescu, R.S., T.M. Mitchell, and R.B. Rao, *Bayesian Network Learning with Parameter Constraints.* J. Mach. Learn. Res., **7**: 1357-1383. 2006.

[21] Cheng, B.H.C., et al. *Using Security Patterns to Model and analyze security requirements*. In *Requirements for high-assurance systems workshop (RHAS'03)*. 2003

[22] Cuppens, F. and A. Miège. *Modelling Contexts in the Or-BAC Model*. In *19th Annual Computer Security Applications Conference (ACSAC '03)*. 2003

[23] Kalam, A.A.E., et al. *Organization Based Access Control*. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*. 2003

[24] Ko, C., M. Ruschitzka, and K. Levitt. *Execution monitoring of security-critical programs in distributed systems: a Specification-based approach*. In *SP '97: IEEE Symposium on Security and Privacy*. 175-187. 1997

[25] Chari, S.N. and P.-C. Cheng, *Bluebox: a policy-driven, host-based intrusion detection system.* ACM Trans. Inf. Syst. Secur., **6**(2): 173-200. 2003.

[26] Ilgun, K., R.A. Kemmerer, and P.A. Porras, *State transition analysis : a rule-based intrusion detection system.* IEEE Trans. Software Eng., **21**(3): 191-199. 1995.

[27] Kumar, S. and E.H. Spafford. *A Pattern Matching Model for Misuse Intrusion Detection*. In *17th National Computer Security Conference*. 11-21. 1994

[28] Valdes, A. and K. Skinner. *Adaptive, Model-based Monitoring for Cyber Attack Detection*. In *Recent Advances in Intrusion Detection (RAID 2000)*. 80-92. Springer. LNCS. 2000

[29] Lamsweerde, A.v., et al. *From system goals to intruder Anti-goals: attack generation and resolution for security requirements engineering*. In *Requirements for high-assurance systems workshop (RHAS'03)*. 2003