Data Sci. Eng. (2016) 1(3):139–148
DOI 10.1007/s41019-016-0016-y

Formal Modelling of Data Integration Systems Security Policies

Fatimah Akeel[1,2] · **Asieh Salehi Fathabadi**[1] · **Federica Paci**[1] · **Andrew Gravell**[1] · **Gary Wills**[1]

Received: 27 July 2016 / Revised: 3 August 2016 / Accepted: 5 August 2016 / Published online: 23 August 2016

Abstract Data Integration Systems (DIS) are concerned with integrating data from multiple data sources to resolve user queries. Typically, organisations providing data sources specify security policies that impose stringent requirements on the collection, processing, and disclosure of personal and sensitive data. If the security policies were not correctly enforced by the integration component of DIS, the data is exposed to data leakage threats, e.g. unauthorised disclosure or secondary use of the data. SecureDIS is a framework that helps system designers to mitigate data leakage threats during the early phases of DIS development. SecureDIS provides designers with a set of *informal* guidelines written in natural language to specify and enforce security policies that capture confidentiality, privacy, and trust properties. In this paper, we apply a *formal* approach to model a DIS with the SecureDIS security policies and verify the correctness and consistency of the model. The model can be used as a basis to perform security policies analysis or automatically generate a Java code to enforce those policies within DIS.

✉ Fatimah Akeel
fya1g12@ecs.soton.ac.uk

Asieh Salehi Fathabadi
asf08r@ecs.soton.ac.uk

Federica Paci
f.m.paci@ecs.soton.ac.uk

Andrew Gravell
amg@ecs.soton.ac.uk

Gary Wills
gbw@ecs.soton.ac.uk

[1] University of Southampton, Southampton, UK

[2] King Saud University, Riyadh, Saudi Arabia

Keywords Security policy · Event-B · Formal method · Privacy · Trust · Confidentiality · RBAC · Trust model · Access control · Modelling

1 Introduction

With the advent of cloud computing and big data analysis, Data Integration Systems (DIS) regained popularity. DIS retrieve data from multiple sources to resolve consumer queries [19]. The main architecture of a DIS consists of a *mediator* [22] that provides an interface between *data consumers* and a set of *data sources*. Data consumers place queries that are resolved by the mediator by integrating data from different data sources.

Organisations providing data sources specify the security policies that impose stringent requirements on the collection, processing, and disclosure of personal and sensitive data. Integrating and enforcing these policies is the responsibility of the mediator during the execution of a query placed by a data consumer. However, if the mediator does not correctly enforce the security policies, this can result in serious data leakage and/or privacy violations leading to significant legal and financial consequences.

Data leakage can occur in a DIS by violating the *confidentiality* of data provided by data sources. For example, the queries executed by the mediator expose data to consumers that were not allowed to access that data according to the security policies of the data sources. Moreover, data leakage can occur by violating the *privacy* of the data when a query discloses the data to a consumer that has a purpose different from the data sources allowed purposes. However, even when the mediator enforces security policies on the execution of a query that discloses data only to authorised consumers and only if the purpose of the query matches the purpose for which the data has been collected, data leakage threats can still materialise in a DIS.

In fact, once the data has been disclosed to data consumers, it is possible that the consumer does not process the data according to the data sources' security policies. The consumer may share the data with unauthorised parties or use the data for fraudulent purposes.

Therefore, to mitigate data leakage threats in DIS, it is important to enforce security policies that not only specify who is entitled to access the data and for which purposes, but also take into account the risks of disclosing this data to data consumers. The risks of consumers not behaving according to a security policy are usually quantified by the degree of *trust* [23] placed into the consumer.

SecureDIS [4] is a novel framework to design DIS resilient to data leakage threats. In order to mitigate data leakage threats, SecureDIS argues that it is very important to enforce security policies that satisfy Confidentiality, Privacy, and Trust (CPT) properties. In particular, SecureDIS helps system designers in considering data leakage threats into the early design phases of DIS by mapping data leakage threats to the different components of a DIS architecture [5] and by providing a set of *informal* guidelines, written in natural language, to implement security policies that mitigate those risks.

In this paper, we provide a *formal* approach to model SecureDIS security policies enforced on the execution of a query. The approach consists of modelling the DIS and the SecureDIS security policies, and verifying the consistency and correctness of the model using Event-B formal method [1] supported by Rodin toolset [2]. The generated model can help designers to analyse the policies or to automatically generate a Java code to enforce the policies within DIS.

The rest of the paper is organised as follows: Section 2 provides an overview on the SecureDIS framework and the requirements to specify and enforce security policies that mitigate data leakage threats. Section 3 explains the modelling of the security policies in Event-B. Section 4 discusses the formal verification of the model. Section 5 reviews the related work, while Section 6 concludes and discusses future research directions.

2 An Overview of the SecureDIS Framework

SecureDIS [4] is a design framework that assists system designers in building DIS resilient to data leakage threats. The framework consists of three main components: (a) a reference architecture of the DIS; (b) a list of data leakage threats mapped to DIS architectural components; and (c) a set of guidelines to mitigate data leakage threats. The components are described as follows:

(a) *The reference architecture* it consists of the following components, as shown in Fig. 1:

1. *Data and data sources* is the core component of the DIS representing the data sources integrated, through their data items, to answer consumers' queries.
2. *The integration approach* is the approach or method used to integrate the data.
3. *The integration location* is the location where the integration process takes place to answer data consumers' queries.
4. *Security policies* combine the security policies from different data sources. It is enforced by the integration location during the execution of the queries.
5. *Data consumers* represent the client side of the system, where data consumers request data by queries and where the results are returned to consumers.

(b) *The data leakage threats* SecureDIS considers different types of data leakage threats, such as inference attacks, unauthorised access, secondary use of information, and non-compliance to policies [5]. Each threat is mapped to one or more of the architectural components of the DIS to understand its consequences and the ways to mitigate it. For example, unauthorised disclosure of sensitive data within the integration location to an entity either inside or outside the DIS is caused by the lack of employing data protection techniques.

(c) *The SecureDIS guidelines* Each component of the DIS architecture is associated with a set of data leakage mitigation guidelines. These guidelines represent the activities proposed to system designers, such as the use of security policies, encryption, and logging. Each guideline covers one or more of the CPT properties and targets one or more data leakage threats. For example, SecureDIS suggests logging and analysing consumers' queries at the data consumers component in order to identify possible secondary use threats.

2.1 SecureDIS Guidelines and Requirements for Security Policies

In this work, we focus on the SecureDIS guidelines related to the specification and enforcement of security policies achieved at the integration location. SecureDIS argues that in order to mitigate data leakage threats, it is crucial for security policies to include the following CPT properties:

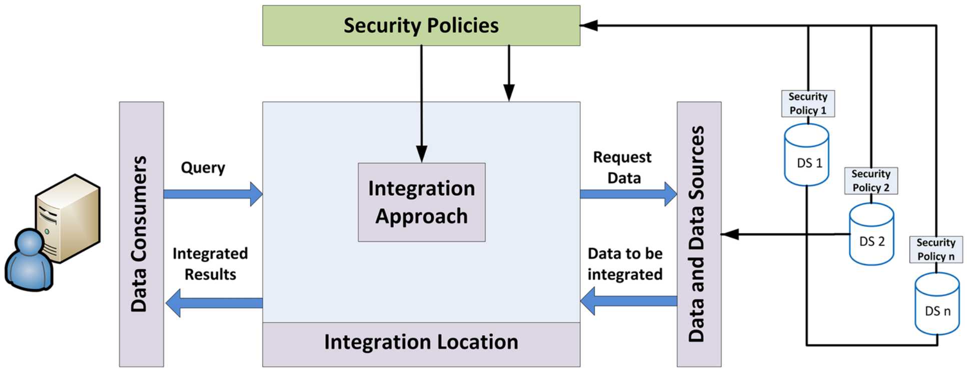

Fig. 1 SecureDIS architecture

Confidentiality is defined as limiting access to authorised entities [17]. Data leakage threats to confidentiality materialise when the integration location returns data items to a consumer who was not allowed to access those items based on the data sources' security policies. To avoid this threat is important to implement a Role-Based Access Control (RBAC) policy and configure it so that each consumer can access only the pieces of data necessary to answer the query [21].

Privacy is the right of the individual to decide what information about himself/herself should be communicated to others and under what circumstances [27]. Data leakage threats to privacy are caused by: disclosing data for purposes different from the one for which the data has been collected [13], by revealing Personally Identifiable Information (PII) intentionally or unintentionally, or by exposing sensitive information protected by data protection laws and regulations. Therefore, in order to prevent data leakage threats to privacy, the security policy should include the following two dimensions:

– *Purpose* determines the reasons for data to be collected or used. The integration location should only grant the execution of a query if the purpose of the query specified by the data consumer matches one of the purposes for which the data items have been collected [13].
– *Data sensitivity* quantifies who should have access to data items and how much harm would be done if the data was disclosed. In order to protect the disclosure of sensitive data items, a security policy similar to the one of the Bell–LaPadula mandatory access control model [12] should be enforced by the data integration location. The security policy should restrict access to data items returned by a query based on the sensitivity of the data items (represented by a label) and the authorisation (represented by security level) of consumers to access data items of such sensitivity. Therefore, the integration location should only grant the execution of a query if the security level assigned to the data consumer is higher or equal to the sensitivity label assigned to data items returned by that query.

Trust is defined as the belief that an entity will behave in a predictable manner by following a security policy [24]. Therefore, trust is used to quantify the risk of data leakage threats that materialise after the execution of queries is granted to data consumers. Once the data is disclosed to data consumers, they could misuse the data by sharing it with unauthorised parties or use it for purposes other than the data provider's intended purposes. Therefore, the security policy should grant the execution of a query only if the trust level of the data consumer is equal or higher to the one specified by the data sources' security policies.

The guidelines above are transformed into specific system requirements shown in Table 1, where the property column indicates the CPT property covered by the requirement. These requirements are used to model the security policies of the DIS.

3 Formal Modelling of Security Policies in Event-B

This section starts by introducing the Event-B formal method, followed by the process of modelling SecureDIS security policies in Event-B.

3.1 Overview of the Event-B Formal Method

Formal methods have been widely used to specify systems rigorously and to ensure the specification is correct and consistent. In the area of computer security, formal methods provide a structured approach for modelling systems using mathematical notations [7] that capture the security policies, system properties, and underlying assumptions. We propose using the Event-B formalism to capture security policies of DIS.

Event-B is a formal method extended from B-Method [1]. It is a state-based method that uses set theory as a main distinctive attribute [6] to model systems for specification and verification purposes. A system can be modelled gradually to reflect its complexity by the use of abstraction

Table 1 System requirements details

Req. no.	System requirement	Property	Type
1	Each data consumer must be assigned to a role to access data sources items	C	Specification
2	Each data source specifies which roles are allowed to access the sources data items	C	Specification
3	A data consumer is granted access to data items returned by a query if the assigned role is an allowed role	C	Enforcement
4	Each data consumer specifies a purpose to access data items	P	Specification
5	Each data item is associated with a purpose for which it was collected	P	Specification
6	A data consumer is granted access to data items returned by a query, if the purpose of the query matches the purpose for which the data items were collected	P	Enforcement
7	Each data item is classified based on its sensitivity	P	Specification
8	Each data consumer is assigned to a security level that specifies the authorisation to access data of a certain sensitivity	P	Enforcement
9	A data consumer is granted access to data items returned by a query, if the security level of the consumer is equal to the sensitivity level of the data items	P	Enforcement
10	Each data consumer is assigned to a trust level	T	Specification
11	Data sources determine the acceptable data consumers trust levels	T	Specification
12	A data consumer is granted access to data items returned by a query, if the trust level of the consumer matches the accepted trust level of data items	T	Enforcement

and refinement techniques. Event-B uses mathematical proofs to ensure the correctness of each level and the consistency between refinement levels [6].

An Event-B model consists of two main components: CONTEXT and MACHINE. The CONTEXT includes the static part of the model that defines SETS, CONSTANTS, and AXIOMS to add constraints on the sets. The MACHINE contains the dynamic part of the model that includes VARIABLES, INVARIANTS, and EVENTS. The VARIABLES specify the states of the system and can be modified by guarded EVENTS. The INVARIANTS specify the constraints on variables, which need to be proved true at any state of the system. The verification of the model demonstrates consistency by ensuring the correctness among all refinements.

The integrated toolsets used to model Event-B is Rodin [2]. The verification process achieved by Rodin includes: (1) model checking: by ProB [20] model checker integrated in Rodin and (2) theorem proving: by generating and proving proof obligations.

3.2 Modelling Security Policies

The security policies modelled in this case study are derived from the aforementioned system requirements in Table 1 that are focused on CPT properties. A security policy consists of the following basic components: a subject, permission(s), and an object [8], and targets a specific property.

The security policies that satisfy the requirements in Table 1 are modelled through Event-B refinements. Three levels of refinements are proposed (see Fig. 2), where each level is represented by a CONTEXT, namely C0, C1, and C2:

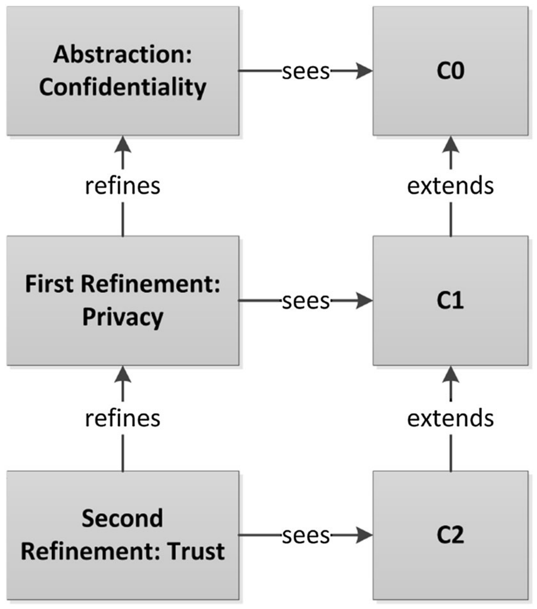

Fig. 2 Security policy refinements

1. *System abstraction* it captures the process of data consumers querying the data provided by different data sources in addition to the security policy that grants the execution of the query. The policy grants the execution only if the consumer is assigned to a specific role that provides the permission to execute the query.
2. *The first refinement* it extends the security policy with the purpose for which data items can be accessed in addition to the data sensitivity.
3. *The second refinement* it extends the security policy with the trust levels that data sources should place into data consumers for granting them the query execution.

3.2.1 System Abstraction: Modelling Confidentiality

The first step is to model the data consumer queries to different data sources and the RBAC policy governing query execution granted to consumers. The system abstraction includes four main sets: *DATA_CONSUMER*, the set of data consumers; *CONSUMER_ROLE*, the set of roles assigned to consumers; *DATA_ITEM*, the set of data items associated with data sources and also returned by queries; and *DATA_SOURCE*, the set of data sources providing the data items to answer data consumers queries.

The system abstraction also includes the main VARIABLES and EVENTS to capture the DIS environment, see Fig. 3. The events are summarised as follows:

- ***AddDataSources*** to add data sources to the model.
- ***AddDataItemsToSources*** to create data items and associate them to data sources.
- ***AddDataConsumers*** to add data consumers to the model.
- ***AddRoles*** to add consumers's roles to the model.
- ***AssignRolesToConsumers*** to assign consumer roles to data consumers.
- ***AddConsumersQueries*** to create consumer queries containing data items.

The variable *belong_to* is defined to associate data items with their data sources, where multiple data items belong to multiple data sources. The invariant that ensures this relation is defined as follows:

inv1 : $belong\_to \in \mathbb{P}1(DATA\_ITEM) \leftrightarrow sources$

Data consumers can access the data items coming from data sources by creating a *query*. The variable *query* is defined as the relationship between consumers and data items. The following invariant shows that multiple consumers can query multiple data items:

inv2 : $query \in consumers \leftrightarrow \mathbb{P}1(DATA\_ITEM)$

However, the *query* has one main restriction that is when a consumer (c) requests a set of data items (*items*), these items need to belong to existing data sources (s). This restriction is enforced by the following invariant:

inv3 : $\forall c, items.c \mapsto items \in query$
$\Rightarrow (\exists s.belong\_to[\{items\}] = s)$

The *query* is created in the ***AddConsumersQueries*** event shown below. The event contains a list of parameters (ANY), a collection of guards (WHERE), and collection of actions (THEN). An event can execute its action(s) only when its guard(s) are true. In this case, the event needs to essentially check whether data items map to sources in **grd4** to satisfy **inv3**.

```
Event AddConsumersQueries
  ANY
    consumer, data_items, source
  WHERE
    grd1 : $consumer \in consumers$
    grd2 : $(data\_items \in \mathbb{P}1(DATA\_ITEM)) \land (data\_items \neq \emptyset)$
    grd3 : $(source \in sources)$
    grd4 : $data\_items \mapsto source \in belong\_to$
  THEN
    act1 : $query := query \cup \{consumer \mapsto data\_items\}$
  END
```

To specify the security policy that captures the confidentiality property, we model the following components:

- The *assigned* invariant to denote that a data consumer can be assigned to more than one role, which fulfils sys. req. 1, and that a role can be assigned to one or more consumers, as follows:

  inv4 : $assigned \in consumers \leftrightarrow roles$

- The *allowed* invariant to indicate the roles allowed to access the data items. Also, *allowed* ensures that data items are actually coming from existing data sources (sys. req. 2). Both these aspects are modelled as follows:

  inv5 : $allowed \in roles \leftrightarrow \mathbb{P}1(DATA\_ITEM)$
  inv6 : $\forall role, items.role \mapsto items \in allowed \Rightarrow$
  $(\exists source.items \mapsto source \in belong\_to)$

- The event ***AddAuthorisation*** to add the RBAC policy to the system by updating the variable *allowed*. To add

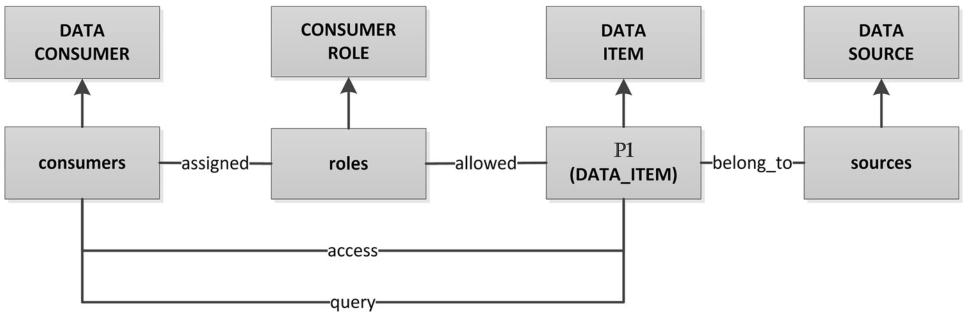

Fig. 3 System abstraction: modelling data query and confidentiality

the pair of a data item (i) and a role (r) to the *allowed* access control list, the guard **grd3** checks whether the data item is associated with existing data sources, as follows:

```
Event AddAuthorisation
    ANY
        $r, i, s$
    WHERE
        grd1 : $i \in \mathbb{P}1(DATA\_ITEM)$
        grd2 : $(s \in sources) \wedge (sources \neq \emptyset)$
        grd3 : $i \mapsto s \in belong\_to$
        grd4 : $(r \in roles) \wedge (roles \notin \emptyset)$
        grd5 : $r \mapsto i \notin allowed$
    THEN
        act1 : $allowed := allowed \cup \{r \mapsto i\}$
    END
```

To model the enforcement of the security policy specified earlier, we include the following:

- **inv7** to model the actual access of consumers to data items and **inv8** to ensure the accessed items are returned by a query:

  inv7 : $access \in consumers \leftrightarrow \mathbb{P}1(DATA\_ITEM)$

  inv8 : $\forall c, items. c \mapsto items \in access \Rightarrow (c \mapsto items \in query)$

- The ***AccessData*** event checks whether the consumer is assigned to a role (**grd3**), and the assigned role is entitled to execute the query (**grd4**), to fulfil sys. req. 3. It also ensures the data items accessed by the consumer are returned as result of a query by the same consumer (**grd2**). The event is modelled as follows:

```
Event AccessData
    ANY
        $consumer, data\_items, consumer\_roles$
    WHERE
        grd1 : $consumer \in consumers$
        grd2 : $data\_items \in query[\{consumer\}]$
        grd3 : $(consumer\_roles \subseteq roles) \wedge$
               $(assigned[\{consumer\}] = consumer\_roles)$
        grd4 : $\exists role.(roles \in consumer\_roles) \wedge$
               $(role \mapsto data\_items \in allowed)$
        grd5 : $(consumer \mapsto data\_items) \notin access$
    THEN
        act1 : $access := access \cup \{consumer \mapsto data\_items\}$
    END
```

3.2.2 First Refinement: Modelling Privacy

The system abstraction in Sect. 3.2.1 models who can have access to the data items returned by a query. This refinement extends the previous level by adding the purpose and data sensitivity privacy dimensions to the security policy as discussed in Sect. 2.1.

To model the *purpose*, we have introduced the following components:

- A set *DATA_USE_PURPOSE* is defined in the context (C1) to include the possible data use purposes assigned to data consumers or data items:

  axm1 : $partition(DATA\_USE\_PURPOSE, \{research\}, \{commercial\}, \{personal\}, \{public\})$

- The variable *item_purpose* is defined to represent the relationship between the $\mathbb{P}1$ (*DATA_ITEM*) and *DATA_USE_PURPOSE*:

  inv9 : $item\_purpose \in \mathbb{P}1(DATA\_ITEM) \leftrightarrow DATA\_USE\_PURPOSE$

- The variable *query_purpose* is defined to represent the relationship between the *consumers* and *DATA_USE_PURPOSE*:

  inv10 : $query\_purpose \in consumers \rightarrow DATA\_USE\_PURPOSE$

- A new event, ***AddItemsPurposes***, to assign several purposes to data items (sys. req. 5). The guards and actions of the event ***AddItemsPurposes*** are as follows:

  grd1 : $purpose \in DATA\_USE\_PURPOSE$

  grd2 : $i \in \mathbb{P}1(DATA\_ITEM)$

  act1 : $item\_purpose := item\_purpose \cup \{i \mapsto purpose\}$

- The event ***AddConsumersQueries*** is refined to assign a purpose to each consumer request to query the system (sys. req. 4) by adding the following:

  grd1 : $purpose \in DATA\_USE\_PURPOSE$

  grd2 : $c \in consumers$

  act1 : $query\_purpose := query\_purpose \cup \{c \mapsto purpose\}$

To model the *data classification*, we include the following:

- A set named *CLASSIFICATION* is defined in the context(C1) to contain the possible levels that can be assigned to data items and data consumers. The set includes the following labels:

- *Regulated* data items that are protected by data protection regulations. For example, the items that contain the PII, such as names, SSN, and credit card numbers. If these items were disclosed, harm is caused to the reputation of the data sources and may lead to financial losses.
- *Confidential* data items that include sensitive information that when disclosed, it can result in a medium level of harm and financial losses.
- *Public* data items that can be disclosed to the general public that when disclosed, it results in a low risk to privacy and reputation.

axm2 : $partition(CLASSIFICATION, \{Regulated\}, \{Confidential\}, \{Public\})$

- A variable *classified* is defined to link each data item with a *CLASSIFICATION* (sys. req. 7) as follows:

inv11 : $classified \in \mathbb{P}1(DATA\_ITEM) \twoheadrightarrow CLASSIFICATION$

- The event ***AddDataItemsToSource*** is refined to classify each data item by updating the variable *classified* as follows:

grd1 : $i \in \mathbb{P}1(DATA\_ITEM)$
grd2 : $j \in CLASSIFICATION$
grd3 : $i \notin dom(classified)$
act1 : $classified := classified \cup \{i \mapsto j\}$

- A variable *security_clearance* to associate a consumer with the security clearance. It is defined as follows:

inv12 : $security\_clearance \in consumers \rightarrow CLASSIFICATION$

- The event ***AddDataConsumers*** is refined to assign each new data consumer an appropriate security clearance (sys. req. 8):

grd1 : $sc \in CLASSIFICATION$
grd2 : $c \in consumers$
grd3 : $c \mapsto sc \notin security\_clearance$
act1 : $security\_clearance := security\_clearance \cup \{c \mapsto sc\}$

To enforce the extended security policy, we refined the *AccessData* event by including the following guards:

- A guard to enforce accessing data when the data consumer's purpose, during query creation, matches one of the data item purposes (sys. req. 6):

grd6 : $item\_purpose[\{data\_items\}] = query\_purpose[\{consumer\}]$

- A guard to ensure that the classification of the data items requested for access matches the consumer's security clearance (sys. req. 9):

grd7 : $security\_clearance[\{consumer\}] = classified[\{data\_items\}]$

3.2.3 Second Refinement: Modelling Trust

The second refinement extends the first to capture the trust property. Trust is introduced into the security policy to minimise threats that are related to secondary disclosure of information caused by data consumers abuse of privileges. Therefore, in this refinement we introduce the trust model proposed in [3]. This trust model labels an entity with any of the following levels: very good, good, neutral, bad, and very bad, based on calculations conducted on that entity to assess its risks. This trust model is included in the second refinement by adding the following components:

- A set *TRUST_LEVEL* containing all possible trust levels in the trust model:

axm3 : $partition(TRUST\_LEVEL, \{very\_good\}, \{good\}, \{neutral\}, \{bad\}, \{very\_bad\})$

- A variable *consumer_tlevel* to associate each data consumer with its trust level:

inv13 : $consumer\_tlevel \in consumers \rightarrow TRUST\_LEVEL$

- A variable *item_tlevel* to associate data items with their acceptable trust levels:

inv14 : $item\_tlevel \in \mathbb{P}1(DATA\_ITEM) \leftrightarrow TRUST\_LEVEL$

- The event ***AddConsumers*** is refined to associate a data consumer with its trust level during the addition of the consumer to the system (sys. req. 10):

grd5 : $c \in consumers$
grd6 : $t \in TRUST\_LEVEL$
act3 : $consumer\_tlevel = consumer\_tlevel \cup \{c \mapsto t\}$

- The event ***AddDataItemsToSources*** is refined to associate data items with acceptable trust levels (sys. 487 req. 11):

$$\begin{aligned}
&\textbf{grd5:} && i \in \mathbb{P}1(DATA\_ITEM)\\
&\textbf{grd6:} && t \in TRUST\_LEVEL\\
&\textbf{act3:} && item\_tlevel = item\_tlevel \cup \{i \mapsto t\}
\end{aligned}$$

To enforce the security policy related to the trust property, we refined the ***AccessData*** event to check whether the consumer's trust level matches the expected trust level associated with the data items returned by a query (sys. req. 12). The following guard is included:

$$\begin{aligned}
\textbf{grd8:}\quad & item\_tlevel[\{data\_items\}]\\
& = consumer\_tlevel[\{consumer\}]
\end{aligned}$$

4 Formal Verification of the Model

The Rodin toolset provides an environment for both modelling and proving by theorem proving and model checking. In addition to formal modelling, we also prove that the proposed Event-B model is correct and consistent. Table 2 presents an overview of the proof efforts provided by Rodin. These statistics measure the proof obligations (PO) generated and discharged by the Rodin prover and the POs that are interactively proved. The complete development of the DIS security policies results in 38 POs, in which (100 %) are proved automatically by Rodin. The number of POs in the system abstraction that captures the confidentiality property is larger than other refinements. This is due to establishing the main components of the security policies (the subject, the permission(s), and the object), and therefore many invariants are introduced in that layer to guarantee the correctness of these components.

4.1 Theorem Proving

There are different POs generated by Rodin during the development of a system [14]. As an example of a PO, we demonstrate an "Invariant Preservation" PO here. The INV PO ensures that each invariant is preserved by each event. To prove that inv 6 , below, is preserved by *AddAuthorisation* event, "AddAuthorisation/inv6/INV" PO is generated and proved by Rodin.

$$\begin{aligned}
\textbf{inv6:}\quad & \forall role, items.role \mapsto items \in allowed \Rightarrow\\
& (\exists source.items \mapsto source \in belong\_to)
\end{aligned}$$

Table 2 The statistics of the model

Element name	Total	Auto	Manual
Model	38	38	0
Confidentiality	25	25	0
Privacy	9	9	0
Trust	4	4	0

To prove this PO, guard **grd3** below is added to the *AddAuthorisation* event to ensure that each role is linked to a data item that actually belongs to a data source.

$$\textbf{grd3:}\quad i \mapsto s \in belong\_to$$

4.2 Model Checking

ProB is an animator and model checker for Event-B. ProB allows fully automatic exploration of Event-B models and can be used to systematically check a specification for a range of errors. We analysed our model using ProB to ensure that the model is deadlock free. For each new event added in the refinements, we have verified that it would not introduce a deadlock using ProB.

5 Related Work

The work presented in this paper is related to two main areas of research: security and privacy engineering and formal analysis of security policies.

5.1 Security and Privacy Engineering

This area of research focuses on considering security and privacy threats in the early phases of the software development lifecycle. Two of the popular techniques that help system designers in identifying security and privacy threats are Microsoft's STRIDE [25] and LINDDUN [9]. STRIDE provides a taxonomy of the type of threats. It is the acronym of: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. Each of these categories of threats negates a security property, namely confidentiality, integrity, availability, authentication, authorisation, and non-repudiation. STRIDE guides the system designers on the identification of security threats through a systematic process. First, a model of the system is created and the system components are mapped to the six threat categories. Then, a catalogue of threat tree patterns is used to identify specific instances of threat categories, where the level of risk of each threat is determined. Finally, the risk of the threat is reduced or eliminated by introducing proper countermeasures and defences.

LINDDUN follows a process similar to STRIDE to help system designers in identifying privacy rather than security threats. Similar to STRIDE, LINDDUN provides a taxonomy of privacy threats that violate specific privacy properties. It includes an extensive catalogue of specific threats. For each category of threats, a list of privacy-enhancing technologies that mitigate privacy threats are provided.

Similar to STRIDE and LINDDUN, SecureDIS framework aims to help designer in identifying threats early in the software development lifecycle. However, SecureDIS focuses only on a specific category of threats, namely data leakage threats, and a specific type of systems, namely DIS.

5.2 Formal Analysis of Security Policies

This area of research focuses on automated methods and tools to detect and correct errors in policy specifications before they are deployed. Several approaches have been proposed to analyse security policies, which mainly differ in the formalism and tools used to model and analyse the policies. These approaches pursue different techniques, ranging from SMT formulae to Multi-Terminal Binary Decision Diagrams (MTBDD) and different kinds of logics.

Margrave [11] uses MTBDDs as the underlying representation of XACML policies. It supports two main types of policy analysis: policy querying, which analyses access requests evaluated to a certain decision, and change-impact analysis, which is used to compare policies. However, BDD-based approaches allow the analysis of policies only against a limited range of properties.

Alternative approaches encode policies and properties as propositional formulas and analyse them using SAT solvers [16]. However, SAT solvers cannot handle Boolean variables and therefore are limited in the type of access control policies that can be modelled and analysed.

Other formalisms have also been used for the analysis of access control policies. Description logic (DL) [18] is used to formalise access control policies and employs off-the-shelf DL reasoners for policy analysis. The use of DL reasoners allows modelling more expressive access control policies, but it suffers from scalability issues.

Answer Set Programming (ASP) [10] has also being used to model and analyse access control policies, but it also has some limitations. ASP does not support quantifiers and does not easily allow the expression of constraints, such as Linear Arithmetic.

More recent approaches to policy analysis are based on SMT [26]. The use of SMT does not only enable wider coverage of access control policies compared to the analysis tools mentioned above but also improves the performance.

Similar to our work, other works have applied Event-B to model and analyse access control policies [6, 15]. One of the main advantages of using Event-B to model and analyse security policies is that it is possible to model not only the access control policies but also the system where the policies are going to be deployed. Another advantage is the expressiveness of the Event-B formalism that allows modelling fine-grained policies. Last, but not least, the Rodin tool that supports the Event-B formalism allows Java code generation from the formal model of the system and its policies.

6 Conclusion and Future Work

SecureDIS is a framework that helps system designers to mitigate data leakage threats during the early phases of the DIS development. SecureDIS provides designers with a set of *informal* guidelines written in natural language to specify and enforce security policies that capture Confidentiality, Privacy, and Trust (CPT) properties.

In this paper, we applied a *formal* approach to model the SecureDIS system and its security policies and verify the correctness and consistency of the model. We used Event-B formal method to formalise the requirements on the specific policy elements that satisfy the CPT properties. These elements were gradually built throughout the model by utilising Event-B abstraction and refinements.

Modelling security policies that capture the SecureDIS main properties is useful to demonstrate how access to data can be controlled by several conditions, as explained in Sect. 3: the allowed role specified as invariant 5 and 6, the allowed purpose specified as guard 6 of the AccessData Event, and the allowed trust level specified as guard 7 of the AccessData Event. This helps in mitigating the threats of data leakage by minimising data exposure due to the incorrect specification of the security policies, such as unauthorised access, non-compliance to security policy, and the misuse of data by authorised consumers.

We are planning to extend this work in several directions. A first direction is to model an instance of a real DIS along with its security policies and check the correctness of those policies before deployment. Another direction is to use the correct model of the security policies to automatically generate the code for their enforcement using the Rodin tool.

Acknowledgments The work reported in this paper is funded by King Saud University, Riyadh, Saudi Arabia.

References

1. Abrial JR (2010) Modeling in Event-B: system and software engineering. Cambridge University Press, Cambridge
2. Abrial JR, Butler M, Hallerstede S, Voisin L (2006) An open extensible tool environment for Event-B. Formal methods and software engineering. Lect Notes Comput Sci 4260:588–605

3. Agudo I, Fernandez-Gago C, Lopez J (2010) A scale based trust model for multi-context environments. Comput Math Appl 60(2):209–216
4. Akeel F, Wills G, Gravell A (2013) SecureDIS: a framework for secure data integration systems. In: The 8th international conference for internet technology and secured transactions, pp 588–593
5. Akeel FY, Wills GB, Gravell AM (2014) Exposing data leakage in Data Integration Systems. In: 9th International conference for internet technology and secured transactions, ICITST 2014, pp 420–425
6. Butler M (2013) Mastering system analysis and design through abstraction and refinement. In: Broy M, Peled D, Kalus G (eds) Engineering dependable software systems. IOS Press, pp 49–78
7. Butler MJ, Leuschel M, Presti SL, Turner P (2004) The use of formal methods in the analysis of trust (position paper). Trust Manag Lect Notes Comput Sci 2995:333–339
8. Crampton J, Huth M (2010) Towards an access-control framework for countering insider threats. Adv Inf Secur 49:173–195
9. Deng M, Wuyts K, Scandariato R, Preneel B, Joosen W (2011) A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. Requir Eng 16(1):3–32
10. Ramli CDPK, Nielson HR, Nielson F (2013) XACML 3.0 in Answer Set Programming. In: Logic-based program synthesis and transformation. Springer, Berlin, pp 89–105
11. Fisler K, Krishnamurthi S, Meyerovich LA, Tschantz MC (2005) Verification and change-impact analysis of access-control policies. In: Proceedings of the 27th international conference on software engineering, pp 196–205
12. Gollmann D (1999) Computer security. Wiley, New York
13. Guarda P, Zannone N (2009) Towards the development of privacy-aware systems. Inf Softw Technol 51(2):337–350
14. Hallerstede S (2011) On the purpose of Event-B proof obligations. Form Asp Comput 23(1):133–150
15. Hoang TS, Basin D, Abrial JR (2009) Specifying access control in Event-B. Technical report, vol 624
16. Hughes G, Bultan T (2008) Automated verification of access control policies using a SAT solver. Int J Softw Tools Technol Transf 10(6):503–520
17. ISO: ISO/IEC27000 (2014) Information technology: security techniques: information security management systems: overview and vocabulary
18. Kolovski V, Hendler J, Parsia B (2007) Analyzing web access control policies. In: Proceedings of the 16th international conference on World Wide Web—WWW '07, p 677
19. Lenzerini M (2002) Data integration: a theoretical perspective. In: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. Madison, Wisconsin, USA, pp 233–246
20. Leuschel M, Butler M (2003) The ProB animator and model checker for B - A tool description. Int Symp Form Methods Eur 2805:855–874
21. McCallister E, Grance T, Scarfone K (2010) Guide to protect the confidentiality of personal identifiable information (PII). NIST Special Publication (800-122), p 59
22. Nachouki G, Quafafou M (2011) MashUp web data sources and services based on semantic queries. Inf Syst 36(2):151–173
23. Paci F, Fernandez-Gago C, Moyano F (2013) Detecting insider threats: a trust-aware framework. In: 2013 Eighth international conference on availability, reliability and security (ARES), pp 121–130
24. Ross R, Oren JC, Mcevilley M (2014) Systems security engineering an integrated approach to building trustworthy resilient systems. NIST Special Publication (800-160), p 121
25. Torr P (2005) Demystifying the threat modeling process. IEEE Secur Priv 3(5):66–70
26. Turkmen F, Den Hartog J, Ranise S, Zannone N (2015) Analysis of XACML policies with SMT. Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 9036, pp 115–134
27. Westin A (1970) Privacy and freedom. Bodley Head, London