Maintaining Information Flow Security Under Refinement and Transformation

Fredrik Seehusen[1,2] and Ketil Stølen[1,2]

[1] SINTEF Information and Communication Technology, Norway
[2] University of Oslo, Norway
{fse,kst}@sintef.no

Abstract. We address the problem of maintaining information flow security under refinement and transformation. To this end we define a schema for the specification of secure information flow properties and show that all security properties defined in the schema are preserved by a notion of refinement. Refinement is a process that requires human guidance and is in general not subject for automation. A transformation on the other hand, is an executable function mapping specifications to specifications. We define an interpretation of transformations and propose a condition under which transformations maintain security.

1 Introduction

We address the problem of maintaining information flow security during the process of making an abstract specification more concrete. This problem has received little attention, yet it is of relevance in any real-life scenario in which security analysis is carried out on the basis of a specification that abstracts away details of the full implementation. For example, it is of little help to know that Java code or some state machine specification is secure w.r.t. some property if validity of the property is not maintained by the compiler. Hence, we need means and a corresponding theory to ensure that the transformation from the abstract level to the more concrete level *maintains* the security of the abstract level.

Proving security once and for all is in general not possible. One reason for this is that the concrete level often includes peculiarities that do not have any abstract equivalent. Consequently security must be proven again at the concrete level to ensure that these additional peculiarities introduced via transformation do not violate security. Although additional verification is often needed at the concrete level, we still want to check and maintain security properties on the basis of the abstract specifications. There are three main reasons for this. First, analysis is in general more feasible at the abstract level since the concrete level may include too much detail to make analysis practical. Second, abstract specifications are in general more platform independent than concrete specifications. This means that analysis results are more reusable at the abstract levels. Third, abstract specifications tend to be more understandable than concrete specifications, hence it is in general easier to specify and check security requirements at abstract levels as opposed to the concrete levels.

In this paper we consider security in the sense of *secure information flow* properties (see e.g. [4,5,16]). The notion of secure information flow provides a way of specifying security requirements by selecting a set of observers, i.e. abstractions of system entities, and then restricting allowed flow of information between the observers.

The process of making an abstract specification more detailed is known as *refinement*, and the relationship between information flow security and refinement has been researched for a fairly long time. In 1989 it was shown by Jacob [13] that secure information flow properties in general are not preserved by the standard notion of refinement. It has later been observed that the problem originates in the inability of most specification languages to distinguish between underspecification and unpredictability[1] [10,14,19]. We argue that this distinction is essential if secure information flow properties are to be preserved under refinement. To this end, both the standard notion of refinement and all secure information flow properties proposed in literature have to be redefined such that this distinction is taken into consideration. We show how to do this in a formalism similar to STAIRS [7,8,9] by defining a schema (based on [16]) for specifying secure information flow properties such that all properties defined in the schema are preserved by refinement.

Refinement is a relation on specifications that formalizes the process of stepwise development by the removal of underspecification. A transformation on the other hand is a computable function mapping specifications to specifications. For example, a compiler mapping a program to machine code is a kind of transformation. Currently, there is much ongoing work on transformation in relation to OMG's standardization activities on MDA (Model Driven Architecture) [18], where transformations characterize the mapping of PIM's (Platform Independent Model) to PSM's (Platform Specific Model). Motivated by this we give a semantic interpretation of transformations and propose a condition under which transformations maintain security.

In summary, the main contributions of this paper are: (1) the definition of a schema for specifying secure information flow properties that are preserved by the STAIRS notion of refinement. (2) The definition of a notion of secure transformation that preserves security properties defined in our schema.

This paper is structured as follows: Sect. 2 formalizes a notion of *system specification*. Sect. 3 describes what is meant by secure information flow. In Sect. 4 we present the STAIRS notion of refinement and propose a schema for specifying secure information flow properties that are preserved by this notion of refinement. In Sect. 5, we discuss security maintaining transformations. Sect. 6 provides conclusions and related work.

2 System Specifications

We model the input-output behavior of systems by finite sequences of *events* called *traces*. An event represents the *transmission* or the *reception* of a message. Formally, an event is a pair (k, m) consisting of a kind k and a message m. An event whose kind equals ! represents the transmission of a message, whereas an event whose kind equals ? represents the reception of a message. A message is a triple (a_1, a_2, s) consisting of a transmitter a_1, a receiver a_2, and a signal s representing the message body. Both

[1] Also termed *probabilistic non-determinism* [19].

transmitters and receivers are referred to as *agents*, i.e. system entities such as objects or components.

Definition 1. *The semantics of a system specification, denoted Φ, is a prefix-closed set of traces. A set of traces A is prefix-closed iff*

$$t \in A \wedge t' \sqsubseteq t \Rightarrow t' \in A$$

where $\sqsubseteq$ is the standard prefix ordering on sequences.

The reason why we require prefix-closure is that the definition of many secure information flow properties proposed in literature rely on this requirement [16,24].

In the sequel, we will for short write "specification" instead "the semantics of system specification" when it clear from the context what is meant.

2.1 Notational Convention

We define some notational conventions and standard operations. $\mathbb{P}(A)$ denotes the power set of A defined $\{X | X \subseteq A\}$. A^* denotes the set of all finite sequences over the set A. A sequence of *events*, i.e. a trace, is written $\langle e_1, e_2, ..., e_n \rangle$. The empty trace, i.e. the trace with no events is written $\langle \rangle$. The *projection* of a trace t on a set of events E, written $t|_E$, is obtained from t by deleting all elements not in E.

Further notational conventions are listed in Table 1. Here the notion $a \in A$ means that the set A is ranged over by a.

Table 1. Notational conventions

Set	Definition	Meaning	Set	Definition	Meaning
$a \in \mathcal{A}$		Set of agents.	$e \in \mathcal{E}$	$\{!, ?\} \times \mathcal{M}$	Set of events.
$o \in \mathcal{O}$	$\mathbb{P}(\mathcal{A})$	Set of observers.	$h \in \mathcal{H}$		Set of high-lvl. events.
$s \in \mathcal{S}$		Set of signals.	$l \in \mathcal{L}$		Set of low-lvl. events.
$m \in \mathcal{M}$	$\mathcal{A} \times \mathcal{A} \times \mathcal{S}$	Set of messages.	$t \in \mathcal{T}$	$\mathcal{E}^*$	Set of traces.

3 Information Flow Security

By secure information flow we understand a restriction on allowed flow of information between *observers*, i.e. sets of agents. Secure information flow can be described by a *flow policy* and a *secure information flow predicate*, referred to as a security predicate for short. The flow policy restricts information flow between observers, while the security predicate defines what is meant by information flow. Formally, a flow policy is a relation on observers

$$\nrightarrow \subseteq \mathcal{O} \times \mathcal{O}$$

where $(o_1, o_2) \in \nrightarrow$ requires that there shall be no information flow from o_1 to o_2.

For simplicity, we will in the sequel assume a fixed flow policy $\{(H, L)\}$ consisting of two observers only: H, the high-level observer and L, the low-level observer.

Security predicates that describe what is meant by information flow are expressed in terms of the observations that observers can make. Formally, the *observation* that an observer o can make of a trace t is obtained from t by deleting all events that can not be observed by o:

$$t|_{(\text{E}.o)}$$

Here $\text{E}.o$ yields the set of all events that can be observed by o, i.e. all events that can be transmitted from or received by the agents in o:

$$\text{E}.o \triangleq \{(k,(a,a',m)) \in \mathcal{E} \mid (k = ! \wedge a \in o) \vee (k = ? \wedge a' \in o)\}$$

For short, we let $\mathcal{L}$ and $\mathcal{H}$ denote $\text{E}.L$ and $\text{E}.H$, respectively.

To ensure that L cannot observe high-level events directly, we demand that $\mathcal{H} \cap \mathcal{L} = \emptyset$. This alone does not in general prevent information flow from H to L because L may *infer* confidential information from H based on the observations that L can make. A central notion in defining what is meant by inferences is that of *low-level indistinguishability*.

Definition 2. *Two traces t and t' are indistinguishable from L's point of view, written $t \sim_{\mathfrak{l}} t'$, iff*

$$t|_{\mathcal{L}} = t'|_{\mathcal{L}}$$

That is, iff L's observation of t is equal to L's observation of t'.

In the sequel we assume that L has complete knowledge of the specification Φ that describes the set of all possible behaviors represented by traces. This means that L may construct the set of all traces in Φ that are indistinguishable or compatible with a given observation. Formally, L may from the observation of any trace t, construct a so-called *low-level equivalence set* [24] (abbreviated LLES in the sequel):

$$\{t' \in \Phi \mid t \sim_{\mathfrak{l}} t'\}$$

In other words, if L makes an observation t, then L can infer that some trace in the LLES constructed from t has occurred, but not which one. Security predicates must demand that L shall not be able to deduce confidential information from the LLESs that L may construct. This is illustrated in the following example.

Example 1. Let $\Phi = \{\langle\rangle, \langle l_1\rangle, \langle h_1\rangle, \langle h_2\rangle, \langle h_1, l_1\rangle, \langle h_1, l_2\rangle, \langle h_2, l_2\rangle\}$, and assume that L may observe events l_1 and l_2 and that h_1 and h_2 are high-level events. Assume further a definition of security that states that L shall not with certainty be able to infer that a high-level event has occurred. If L makes the observation $\langle l_1\rangle$, then L may infer that either trace $\langle l_1\rangle$ or trace $\langle h_1, l_1\rangle$ have occurred. Since L cannot know which of these has occurred (because L cannot observe high-level events directly), and the former trace does not contain any high-level events, L cannot infer with certainty that a high-level event has occurred. If L on the other hand observes the trace $\langle l_2\rangle$, then L can infer that $\langle h_1, l_2\rangle$ or $\langle h_2, l_2\rangle$ have occurred. Again, L does not know which of these has occurred, but since both traces contain a high-level event and there are no other traces in Φ that are compatible with L's observation, L can infer with certainty that a high-level event has occurred. Hence, Φ is not secure w.r.t. our definition of security.

In order for Φ to be secure, one must demand that the LLESs that L can construct be *closed* w.r.t. some criterion [16]. In the above example, this amounts to demanding that there must be a trace with no high-level events in each LLES that L can construct.

3.1 Mantel's Assembly Kit

The schema we propose for describing security predicates is based on a schema proposed by Mantel [16]. He presents an assembly kit in which different notions of security can be defined. We give here a brief description of this assembly kit. The reader is referred to [16] for a more details.

In Mantel's framework, security properties are represented as *security predicates* where a security predicate $\textsc{Sp}$ is either a single *basic security predicate* $\textsc{Bsp}$, or a conjunction of basic security predicates. Each basic security predicate $\textsc{Bsp}$ demands that for any trace t of the specification Φ there must be another trace t' that is indistinguishable from t from L's point of view, and which fulfills a condition Q, the *closure requirement* of $\textsc{Bsp}$. The existence of t', however, is only required if a condition R, the *restriction* of $\textsc{Bsp}$, holds. This results in the following schema for the formal definition of basic security predicates:

Definition 3. *Specification Φ satisfies the basic security predicate $\textsc{Bsp}_{QR}(\Phi)$ for restriction R and closure requirement Q iff*

$$\forall t \in \Phi \cdot R(\Phi, t) \Rightarrow \exists t' \in \Phi \cdot t \sim_{\mathfrak{L}} t' \wedge Q(t, t') \tag{1}$$

Example 2. The notion of security that is informally described in Ex. 1, may be defined by instantiating the schema as follows: $R \triangleq \textsc{true}$, and $Q \triangleq t'|_{\mathcal{H}} = \langle\rangle$. That is, for every trace t there must be a trace t' such that t' is indistinguishable from t w.r.t. L and such that t' does not contain any high-level events.

4 Refinement

Refinement is the process of making an abstract specification more concrete by removing underspecification. The standard notion of refinement [11] states that a system specification Φ' is a refinement of a system specification Φ iff

$$\Phi' \subseteq \Phi \tag{2}$$

Intuitively, there are at least as many implementations that satisfy Φ as there are implementations that satisfy Φ'. In this sense Φ' describes its set of implementations more accurately than Φ, ergo Φ' is less abstract than Φ.

The reason why secure information flow properties are not preserved by refinement becomes apparent when one considers again the manner in which these properties are defined (see Def. 3). That is, Φ is secure if some of its traces satisfy the closure requirement Q. However, by (2) there is no guarantee that a refinement of Φ will include those traces that satisfy Q, hence secure information flow properties are in general not preserved by refinement.

Intuitively, the cause of this problem is that security properties depend on unpredictability. E.g. the strength of ones password may be measured in terms of how hard it is for an attacker to guess the password one has chosen. The closure requirement Q may be seen as the security predicate's requirement of unpredictability, but traces that provide this unpredictability may be removed during refinement. This motivates a redefinition of the notions of specification and refinement where the distinction between underspecification and unpredictability is taken into consideration.

Definition 4. *A system specification, denoted Ω, is a set of trace sets. Each trace set in a specification is called an obligation. We demand that the set obtained by collapsing Ω into a set of traces must be prefix-closed, i.e. we demand*

$$t \in \widehat{\Omega} \wedge t' \sqsubseteq t \Rightarrow t' \in \widehat{\Omega}$$

where $\widehat{\Omega}$ is defined $\bigcup_{\phi \in \Omega} \phi$.

Definition 5. *System specification Ω' is a refinement of system specification Ω, written $\Omega \rightsquigarrow \Omega'$, iff*

$$(\forall \phi \in \Omega \cdot \exists \phi' \in \Omega' \cdot \phi' \subseteq \phi) \wedge (\forall \phi' \in \Omega' \cdot \exists \phi \in \Omega \cdot \phi' \subseteq \phi)$$

This corresponds to so-called limited refinement in STAIRS [20]. For an arbitrary obligation ϕ at the abstract level, there must be an obligation ϕ' at the concrete level such that ϕ' is a refinement of ϕ in the sense of the standard notion of refinement (see (2)). Moreover, each obligation at the concrete level must be a refinement of an obligation at the abstract level. The latter ensures that behavior that was not considered at the abstract level is not introduced at the concrete level.

The intuition is that the traces within the same obligation may provide underspecification, while the obligations provide unpredictability in the sense that an implementation is required to fulfill all obligations of a specification. Any valid implementation must potentially exhibit the behavior described by at least one trace in each obligation. By implementation we understand a specification with no underspecification. Given some program P, let $Traces(P)$ be the prefix-closed set of traces that can be generated by executing P, and let

$$[\![\, P \,]\!] \triangleq \{\{t\} \mid t \in Traces(P)\}$$

Then P implements specification Ω iff

$$\Omega \rightsquigarrow [\![\, P \,]\!]$$

Example 3. Let $\Omega = \{\{\langle\rangle\}, \{\langle l \rangle\}, \{\langle l \rangle, \langle h_1 \rangle, \langle h_2 \rangle, \langle h_1, l \rangle, \langle h_2, l \rangle\}\}$, and assume that it is confidential that high-level events have occurred. Ω is secure in this respect; it is easy to verify that this holds for all implementations of Ω.

Lemma 1. $\rightsquigarrow$ *is transitive*[2]*:*

$$\Omega \rightsquigarrow \Omega' \wedge \Omega' \rightsquigarrow \Omega'' \Rightarrow \Omega \rightsquigarrow \Omega''$$

[2] The proofs of all the results in this paper can be in [22].

Instances of the schema of Def. 3 are in general not preserved by our notion of refinement. We need to modify the schema such that the distinction of unpredictability and underspecification is exploited. Instead of demanding that there is a trace t' that satisfies some criterion, we demand that there is an *obligation* ϕ such that all its traces satisfy that criterion.

Definition 6. *Specification Ω satisfies the basic security predicate $\textsc{Bsp}_{QR}(\Omega)$ for restriction R and closure requirement Q iff*

$$\forall t \in \widehat{\Omega} \cdot R(\widehat{\Omega}, t) \Rightarrow \exists \phi \in \Omega \cdot \forall t' \in \phi \cdot t \sim_{\mathfrak{l}} t' \wedge Q(t, t')$$

The intuition of (Def. 6) is that obligations, as opposed to individual traces, may be seen as providing the unpredictability required by instances of the schema. Note that the schema may be instantiated by the same instances of R and Q that are presented in Mantel's paper.

In order to ensure that instances of the schema are preserved by refinement, we need to disallow some restrictions R whose truth value depend on the absence of traces. We therefore require that all restrictions R satisfy the following condition

$$(T' \subseteq T \wedge R(T', t)) \Rightarrow R(T, t) \tag{3}$$

for arbitrary traces t and trace sets T and T'. All instances of R presented in Mantel's paper satisfy condition (3).

Theorem 1. $\textsc{Bsp}_{QR}$ *is preserved by refinement for arbitrary restrictions R satisfying (3) and closure requirements Q:*

$$\Omega \rightsquigarrow \Omega' \wedge \textsc{Bsp}_{QR}(\Omega) \Rightarrow \textsc{Bsp}_{QR}(\Omega')$$

The notion of refinement introduced above corresponds to what is often referred to as property refinement or behavioral refinement [2]. Property refinement does not capture change in data-structure, i.e. the replacement of abstract event representations by concrete event representations. This is in contrast to refinement notions such as data refinement [12], interface refinement [2], or action refinement [23] which roughly speaking may be understood as property refinement modulo a translation between the concrete and the abstract data structure. Our notion of property refinement may be generalized into a notion of action refinement (actually event refinement in our case) using upwards and downwards simulation [3,12] in a fairly standard manner. To characterize under which conditions this notion of refinement is security preserving is, however, far from trivial. In the following, attention is restricted to a special case of this problem, namely under which conditions *transformations* are security preserving. A transformation may be understood a special case of action refinement where the concrete specification is generated automatically from the abstract specification.

5 Transformation

The notion of refinement addressed above is a binary relation on specifications that formalizes the process of stepwise development by removal of underspecification. A

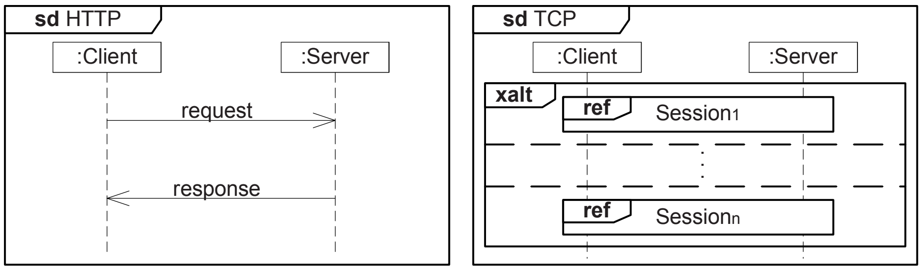

Fig. 1. HTTP to TCP

transformation on the other hand is an executable function taking an abstract syntactic specification as input and yielding a concrete syntactic specification as output. Thus transformation is a syntactic notion. Since security properties are defined on the semantics of specifications (i.e. on traces), we define a semantic interpretation of transformations which enables us to assert whether a transformation maintains security.

In Sect. 5.1, we give an example of a transformation that motivates our semantic interpretation of transformations given in Sect. 5.2. In Sect. 5.3, we propose a condition under which interpretations of transformations maintain security. Sect. 5.4 gives an example that clarifies some of the points made in Sect. 5.3.

5.1 Example: Transforming HTTP Specifications to TCP Specifications

The HTTP protocol is bound to the TCP protocol. One way of doing this binding during runtime is to create a new so-called TCP session for each HTTP request-response pair. A TCP-session consists of three phases: First a connection is established between the two sides of communication, then the HTTP request and response messages are segmented, encapsulated by TCP frames, and transmitted. Finally the connection is explicitly terminated.

A transformation that takes a specification that describes communication at the HTTP level and produces a specification that describes communication at the TCP level may be defined in accordance to how the HTTP protocol is bound to the TCP protocol. Such a transformation may be regarded as a transformation from the abstract to the concrete if one takes HTTP specifications as being at the abstract level and TCP specifications as being at the concrete level.

The UML interaction diagram on the left hand side of Fig. 1 describes a simple communication scenario at the HTTP level. The diagram on the right hand side is the result of applying a transformation from HTTP to TCP to the HTTP diagram. Here the so-called **xalt**-operator from STAIRS [8] specifies unpredictability[3] between different interaction scenarios. The **ref**-operator references other interaction diagrams that in this example describe different TCP-sessions. The reason why the HTTP request-response pair described in the diagram on the left hand side is translated to more than one TCP-session is that the TCP protocol must handle issues that are transparent at the HTTP

[3] Termed explicit non-deterministic choice in STAIRS.

level, e.g. message overtaking. One TCP session may for example describe the situation in which messages are received in the same order that they are transmitted, another may describe the situation in which this is not the case and so on. The reader is referred to [7,8,9,21] to see how UML interaction diagrams can be given trace semantics.

To assert whether the transformation from HTTP to TCP maintains security, we need to interpret the transformation in terms of how abstract traces (representing HTTP communication) are translated to concrete traces (representing TCP communication). There are three considerations that need to be taken into account when defining such an interpretation. First, an abstract trace may correspond to several concrete traces. The reasons for this is that TCP protocol must handle issues that are transparent at the HTTP level. Second, an abstract event may be decomposed into several concrete events because each HTTP package may be segmented into more than one TCP package during the TCP transmission phase. Third, there may be traces at the concrete level that have no abstract equivalent. To see this, let $\langle e \rangle$ represent the transmission of a HTTP package. Since a HTTP package may be segmented into several TCP packages, $\langle e \rangle$ may for example be translated into the trace $\langle e_1, e_2, e_3 \rangle$ where events e_1, e_2, and e_3 represent the transmission of TCP packages. Traces $\langle e_1 \rangle$ and $\langle e_1, e_2 \rangle$ may also be valid traces at the TCP level (these traces are in fact required to be present in a concrete specification since we assume that specifications are prefix-closed). Now, the TCP trace $\langle e_1, e_2, e_3 \rangle$ corresponds to $\langle e \rangle$ at the HTTP level since the TCP trace is *complete* in the sense that it represents the transmission of the entire HTTP message. But what about the TCP traces $\langle e_1 \rangle$ and $\langle e_1, e_2 \rangle$, do these traces correspond to $\langle e \rangle$ at the abstract level? The answer is no if the trace $\langle e \rangle$ is meant to represent the transmission of an entire HTTP package. The TCP traces $\langle e_1 \rangle$ and $\langle e_1, e_2 \rangle$ do not correspond to the empty HTTP trace ($\langle \rangle$) either, because the empty trace is meant (by any reasonable interpretation) to represent a scenario in which no communication occurs. From this we can conclude that, in general, there may be traces at the concrete level for which there are no corresponding traces at the abstract level.

5.2 Transformations from a Semantic Perspective

Syntactically, a transformation is an executable function translating abstract (syntactic) specifications to concrete (syntactic) specifications. Semantically, we interpret traces in terms of how abstract traces are translated to concrete traces.

In the following, let A denote some fixed but arbitrary abstract syntactic specification, T be a some transformation, and $T(A)$ denote the concrete specification obtained by applying T to A. Let Ω_a denote the semantics of A and Ω_c denote the semantics of $T(A)$. T is interpreted by a set of functions

$$F \subseteq \widehat{\Omega}_a \rightarrow \widehat{\Omega}_c$$

mapping traces in Ω_a to traces in Ω_c. The reason why we use a set of functions and not a single function is, as explained in the example, that a syntactic transformation may represent the same abstract trace by several concrete traces.

We say that the set of functions F is a *valid interpretation* of T w.r.t. A if Ω_c is a *translation* of Ω_a w.r.t. F as defined below. We first define the notion of translation for obligations, then we lift this notion to (semantic) specifications.

Definition 7. *Obligation ϕ_c is a translation of obligation ϕ_a w.r.t. function f, written $\phi_a \hookrightarrow_f \phi_c$, iff*

$$\phi_c \subseteq \{f(t) \,|\, t \in \phi_a\}$$

Definition 8. *Specification Ω_c is a translation of specification Ω_a w.r.t. interpretation F, written, $\Omega_a \hookrightarrow_F \Omega_c$, iff*

$$\forall f \in F \cdot \forall \phi_a \in \Omega_a \cdot \exists \phi_c \in \Omega_c \cdot \phi_a \hookrightarrow_f \phi_c$$

Our interpretation of transformations is similar to data refinement in that both notions roughly speaking may be understood as refinement modulo a translation of traces. More precisely:

Lemma 2. *Let Ω_c be contained in the image of Ω_a under the identity transformation id, then*

$$\Omega_a \hookrightarrow_{id} \Omega_c \Leftrightarrow \Omega_a \rightsquigarrow \Omega_c$$

Here $im(\Omega_a, F)$, the *image* of Ω_a under some F, is the set of obligations that are translations of obligations in Ω_a w.r.t. F:

$$im(\Omega_a, F) \triangleq \{\phi_c \,|\, \exists \phi_a \in \Omega_a \cdot \exists f \in F \cdot \phi_a \hookrightarrow_f \phi_c\} \tag{4}$$

A concrete specification is not necessarily contained in the image of the abstract specification it is translated from. The reason for this is, as explained in the previous example, that there may be concrete traces that do not have any abstract equivalent.

If F_1 and F_2 are interpretations, then $F_1 \circ F_2$ is understood as the interpretation obtained by functional point-to-point composition of all functions from F_1 and F_2. That is,

$$F_1 \circ F_2 \triangleq \{f_1 \circ f_2 \,|\, f_1 \in F_1 \wedge f_2 \in F_2\} \tag{5}$$

where $f_1 \circ f_2(t) = f_1(f_2(t))$.

Lemma 3. *$\hookrightarrow$ is transitive:*

$$\Omega_0 \hookrightarrow_{F_1} \Omega_1 \wedge \Omega_1 \hookrightarrow_{F_2} \Omega_2 \Rightarrow \Omega_0 \hookrightarrow_{F_2 \circ F_1} \Omega_2$$

We denote by $F^{-1}(t_c)$, the set of all traces in $\widehat{\Omega}_a$ that can be translated to t_c by the functions in F:

$$F^{-1}(t_c) \triangleq \{t_a \in \widehat{\Omega}_a \,|\, \exists f \in F \cdot f(t_a) = t_c\} \tag{6}$$

5.3 Secure Transformations

A transformation T maintains the security of an abstract specification A if there is a *valid secure interpretation* of T w.r.t. A. The notion of secure interpretation obviously depends on what is meant by secure, and should therefore be parameterized by security properties. In doing so, one must take into account that the security requirement at the abstract level may be syntactically and semantically different from the "corresponding" security requirement at the concrete level. One reason for this is that events at the abstract level may differ from the events at the concrete level. Another reason is that there may be concrete traces that do not have any corresponding abstract trace. The concrete security requirement must therefore handle traces that may not be taken into account by the security requirement at the abstract level.

The notion of secure interpretation is formally defined in the following

Definition 9. *Let $\Omega_a \hookrightarrow_F \Omega_c$, then the interpretation F is secure w.r.t. the abstract and concrete restrictions R_a and R_c and abstract and concrete closure requirements Q_a and Q_c if the following conditions are satisfied*

$$R_c(\widehat{\Omega}_c, t_c) \Rightarrow \exists t' \in F^{-1}(t_c) \cdot R_a(\widehat{\Omega}_a, t') \tag{7}$$

$$\begin{gathered}(R_c(\widehat{\Omega}_c, t_c) \wedge \forall t' \in \phi_a \cdot t_a \sim_{\mathfrak{l}} t' \wedge Q_a(t_a, t')) \Rightarrow \\ \exists \phi_c \in \Omega_c \cdot \forall t' \in \phi_c \cdot t_c \sim_{\mathfrak{l}} t' \wedge Q_c(t_c, t')\end{gathered} \tag{8}$$

for all $t_c \in \widehat{\Omega}_c$, $\phi_a \in \Omega_a$, and $t_a \in F^{-1}(t_c)$.

Def. 9 may be understood to capture a rule that allows us to exploit that we have established Q_a at the abstract level when establishing Q_c at the concrete level. We believe that verifying (7) and (8) in most practical situations will be straightforward and more feasible than checking the security property at the concrete level directly.

Put simply, the first condition of Def. 9 just ensures that the transformation does not weaken the restriction R. The second condition ensures that the transformation does not strengthen the closure requirement Q and that low-level equality is preserved.

It follows from (7) that the concrete restriction R_c must filter away (i.e. yield false for) the concrete traces t_c that do not have any corresponding abstract trace. This is reasonable because one cannot take advantage of the fact that the abstract specification is secure when proving that t_c does not compromise security. It may therefore be the case that a new security analysis must be carried out at the concrete level for those traces that do not have an abstract equivalent.

When we relate Def. 9 to rules that describe date refinement in an assumption / guarantee or pre-post setting, we note that weakening/strengthening is the other way around. E.g., when refining a pre-post specification, one may weaken the pre-condition and strengthen the post-condition. The reason is that a pre-post condition is a specification that is refined into another specification while a restriction-closure predicate is a property that has been proved to hold for a specification that is translated to a concrete specification and whose validity should be maintained.

Theorem 2. *Let F be a interpretation that is secure w.r.t. restrictions R_a and R_c, and closure requirements Q_a and Q_c, then F maintains security in the following sense:*

$$\mathrm{BSP}_{Q_a R_a}(\Omega_a) \wedge \Omega_a \hookrightarrow_F \Omega_c \Rightarrow \mathrm{BSP}_{Q_c R_c}(\Omega_c)$$

5.4 Example: Why Security Requirements Change

Let Ω_a be an abstract specification consisting of two clients c_l and c_h that communicate with a server s via the HTTP protocol (see Fig. 2). Assume that c_l, based on its

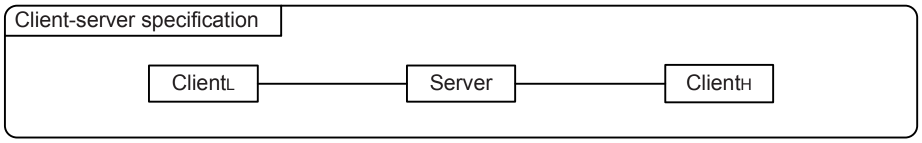

Fig. 2. Client-server example

observation of its communication with s and its knowledge of the system specification, should not be able to deduce information about the behavior of c_h. More formally, both the clients and the server can be represented as agents (recall the role of agents from Sect. 2). Thus the low-level observer is defined $\{c_l\}$ and the high-level observer is defined $\{c_h\}$. The security requirement on the HTTP level may be formalized by instantiating the schema of Def. 6 by some predicates R_a and Q_a.

Assume that F interprets a transformation from HTTP to TCP defined such that each event representing the transmission or reception of a HTTP message is translated into a complete (in the sense of the previous example) sequence of events describing the corresponding TCP messages. Let Ω_c be a translation of Ω_a w.r.t. F and assume that Ω_c also contains non-complete traces that do not correspond to any traces at the HTTP level.

Assume that Ω_a is secure (i.e. $\text{BSP}_{R_a Q_a}(\Omega_a)$ is true) and that we want to check if the traces on the concrete level that have a corresponding abstract representation are secure w.r.t. our information flow property. In order to do this, we can create predicate on the concrete level that filters away those traces that do not have a corresponding abstract representation. More formally, the concrete restriction R_c is defined $R_c(\widehat{\Omega}, t) \triangleq R_a(\widehat{\Omega}, t) \wedge TCP\_OK(t)$ where TCP_OK is a predicate that yields true if the TCP trace t corresponds to a HTTP trace and false otherwise. Note that we are assuming that R_a may take HTTP traces as well as TCP traces as arguments. The concrete security property can now be obtained by instantiating the schema of Def. 6 by the predicates R_c and Q_a (since the closure requirement Q is left unchanged in this example).

If one wants to check that the TCP traces that do not correspond to any HTTP traces are secure w.r.t. some requirement, one cannot take advantage of the fact that Ω_a is secure. Therefore additional security analysis may be required w.r.t. these traces.

6 Conclusions and Related Work

In [21], we defined a secure information flow property in the semantics of STAIRS [7,8,9] and showed that this property was preserved by refinement and transformation. This paper simplifies and generalizes these results by considering, not only one, but many kinds of information flow properties. This paper also considers a more general notion of security preservation than considered in [21]. More precisely, this paper makes two contributions to the study of secure information flow. The first is a schema for specifying information flow properties that are preserved by the STAIRS notion of refinement. The second is the definition of a semantic interpretation of transformations and a condition under which transformations maintain security.

There are a number of papers related to information flow security and refinement. Jacob is the first person that we are aware of to show that secure information flow properties are not preserved by the traditional notion of refinement [13]. This became known as the *refinement paradox*. It has later been observed that this "paradox" is a manifestation of failing to clearly distinguish between underspecification and unpredictability. As far as we are aware of, this observation was first made in [19].

To the extent of our knowledge, the work of Heisel. et. al. [10] is similar to ours in that they both distinguish between underspecification and unpredictability and consider the notion of data refinement. The main differences between their work and ours are: (1) They work in a probabilistic setting, and thus their formalism differs from ours. (2) They do not consider information flow *properties* but a notion of confidentiality based on low-level indistinguishability only. (3) Their notion of confidentiality preserving refinement is different from ours in that they build the condition of confidentiality preservation into the definition of refinement. W.r.t. refinement, we have taken the dual approach of strengthening the notion of security.

The work of Jürjens [14,15] is also related to ours. Some of the main differences between his work and ours are: (1) His formalism differs from ours. (2) While Jürjens distinguishes between underspecification and unpredictability in order to define refinement preserving properties of confidentiality (secrecy) and integrity, he does not rely on this distinction in the definition of his information flow property. That is, the secure information flow property is satisfied iff each behavior refinement to a *deterministic* specification satisfies this property, i.e. he effectively closes the property under a notion of behavior refinement that does not distinguish between underspecification and unpredictability.

Three notable papers that addresses information flow security and refinement are [1,6,17]. The main difference between these papers and ours is that all these investigate conditions under which certain notions of refinement are security preserving without distinguishing between underspecification and unpredictability. Since this distinction is made in our formalism, we consider one notion of refinement only, and strengthen instead our notion of security in an intuitive manner. Hence, there is no need to propose conditions with which to check that a given refinements preserve security.

We are not aware of any work that explicitly address transformation and information flow security. Moreover, we are not aware of any work that show how to preserve secure information flow properties under a notion of refinement that takes the distinction of underspecification and unpredictability into consideration.

The main emphasis of this paper is on semantics. In future work, we will address syntactic transformations in more detail. We are also planning to address composition and transformation of security predicates. Eventually, we would like to develop a computerized tool that will check whether transformations maintain security.

Acknowledgments

This work has been funded by the Research Council of Norway through the project SECURIS (152839/220). We would like to thank the anonymous referees for their useful comments and suggestions.

References

1. Bossi, A., Focardi, R., Piazza, C., Rossi, S.: Refinement operators and information flow security. In: SEFM 2003. 1st International Conference on Software Engineering and Formal Methods, pp. 44–53. IEEE Computer Society Press, Los Alamitos (2003)
2. Broy, M., Stølen, K.: Specification and development of interactive systems. In: FOCUS on streams, interface, and refinement, Springer, Heidelberg (2001)
3. de Roever, W.-P., Engelhardt, K.: Data Refinement: Model-Oriented Proof Methods and their Comparison. In: Cambridge tracts on theoretical computer science, vol. 47, Cambridge University Press, Cambridge (1998)
4. Focardi, R., Gorrieri, R.: Classification of security properties (part i: Information flow). In: Focardi, R., Gorrieri, R. (eds.) Foundations of Security Analysis and Design. LNCS, vol. 2171, pp. 331–396. Springer, Heidelberg (2001)
5. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, pp. 11–20. IEEE Computer Socity Press, Los Alamitos (1982)
6. Graham-Cumming, J., Sanders, J.W.: On the refinement of non-interference. In: Proceedings of the IEEE Computer Security Foundations Workshop, pp. 35–42. IEEE Computer Society Press, Los Alamitos (1991)
7. Haugen, Ø., Husa, K.E., Runde, R.K., Stølen, K.: Why timed sequence diagrams require three-event semantics. Research Report 309, Department of Informatics, University of Oslo (2004)
8. Haugen, Ø., Husa, K.E., Runde, R.K., Stølen, K.: STAIRS towards formal design with sequence diagrams. Journal of Software and Systems Modeling 4(4), 355–367 (2005)
9. Haugen, Ø., Stølen, K.: STAIRS – steps to analyse interactions with refinement semantics. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML2003 - The Unified Modeling Language. Modeling Languages and Applications. LNCS, vol. 2863, pp. 388–402. Springer, Heidelberg (2003)
10. Heisel, M., Pfitzmann, A., Santen, T.: Confidentiality-preserving refinement. In: CSFW-14 2001. 14th IEEE Computer Security Foundations Workshop, pp. 295–306. IEEE Computer Society Press, Los Alamitos (2001)
11. Hoare, C.A.R.: Communicating Sequential Processes. Series in computer science. Prentice-Hall, Englewood Cliffs (1985)
12. Hoare, C.A.R., He, J., Sanders, J.W.: Prespecification in data refinement. Information Processing Letters 25(2), 71–76 (1987)
13. Jacob, J.: On the derivation of secure components. In: Proc. of the IEEE Symposium on Security and Privacy, pp. 242–247. IEEE Computer Society Press, Los Alamitos (1989)
14. Jürjens, J.: Secrecy-preserving refinement. In: Oliveira, J.N., Zave, P. (eds.) FME 2001. LNCS, vol. 2021, pp. 135–152. Springer, Heidelberg (2001)
15. Jürjens, J.: Secure systems development with UML. Springer, Heidelberg (2005)
16. Mantel, H.: Possibilistic definitions of security - an assembly kit. In: CSFW'00. IEEE Compuer Security Foundations Workshop, pp. 185–199. IEEE Computer Society Press, Los Alamitos (2000)
17. Mantel, H.: Preserving information flow properties under refinement. In: IEEE Symposium on Security and Privacy, pp. 78–91. IEEE Computer Society Press, Los Alamitos (2001)
18. Object Management Group.: Architecture Board ORMSC. Model Driven Architecture (MDA). Document number ormsc/2001-07-01 (2001)
19. Roscoe, A.: CSP and determinism in security modelling. In: IEEE Symposium on Security and Privacy, pp. 114–127. IEEE Computer Society Press, Los Alamitos (1995)
20. Runde, R.K., Haugen, Ø., Stølen, K.: Refining uml interactions with underspecification and nondeterminism. Nordic Journal of Computing 12(2), 157–188 (2005)

21. Seehusen, F., Stølen, K.: Information flow property preserving transformation of UML interaction diagrams. In: SACMAT 2006. 11th ACM Symposium on Access Control Models and Technologies, pp. 150–159. ACM Press, New York (2006)
22. Seehusen, F., Stølen, K.: Maintaining information flow security under refinement and transformation. Technical report SINTEF A311, SINTEF ICT (2006)
23. van Glabbeek, R.J., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Informatica 37(4/5), 229–327 (2001)
24. Zakinthinos, A., Lee, E.S.: A general theory of security properties. In: Proc. of the IEEE Computer Society Symposium on Research in Security and Privacy, pp. 94–102. IEEE Computer Society Press, Los Alamitos (1997)