Analyzing Air-traffic Security using GIS-"blur" with Information Flow Control in the IIIf

Florian Kammüller
f.kammueller@mdx.ac.uk
Middlesex University London
London, UK
Technische Universität Berlin
Berlin, Germany

ABSTRACT

In this paper, we address security and privacy of air-traffic control systems. Classically these systems are closed proprietary systems. However, air-traffic monitoring systems like flight-radars are decentralized public applications risking loss of confidential information thereby creating security and privacy risks. We propose the use of the Isabelle Insider and Infrastructure framework (IIIf) to alleviate the security specification and verification of air traffic control systems. This paper summarizes the IIIf and then illustrates the use of the framework on the application of a flight path monitoring system. Using the idea of blurring visual data to obfuscate privacy critical data used in GIS systems, we observe that for dynamic systems like flightradars, implicit information flows exist. We propose information hiding as a solution. To show the security of this approach, we present the extension of the IIIf by a formal notion of indistinguishability and prove the central noninterference property for the flight path monitoring application with hiding.

CCS CONCEPTS

• **Security and privacy → Software and application security**; • **Human-centered computing → Visualization**; • **Theory of computation → Logic**.

KEYWORDS

Air-traffic control systems, Automated reasoning, Information flow control, Formal modeling and verification, Isabelle

ACM Reference Format:
Florian Kammüller. 2024. Analyzing Air-traffic Security using GIS-"blur" with Information Flow Control in the IIIf. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30–August 02, 2024, Vienna, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3664476.3670928

1 INTRODUCTION

Privacy as well as security may be endangered when there appear daily differences on airplane routes. Such irregularities may reveal secret information, for example, certain areas cannot be overflown because of security alerts, or security or privacy critical passengers are on board of the airplanes so that higher security precautions impose maximally safe airplane routing. These precautions are meant to increase security and privacy by physical measures. However, on the level of confidential information they represent implicit information flows. Geographic Information Systems (GIS) offer the possibility to obfuscate private information by blurring the details of the maps thus establishing some level of privacy. However, this does not scale to dynamic information as needed for example for airtraffic control systems where blurring could cause implicit information flows. In formal techniques for software verification such implicit information flows are well known and countermeasures are well understood in an area called Information Flow Control (IFC). In addition to Access Control, which regulates access to objects, IFC "specifies valid channels along which information may flow" [2]. It serves to secure flows of information in software systems by labeling values with security classes and enforcing the security policy by only allowing information to flow between classes if they are in a flow relation [3]. IFC has become a standard technique in the security specification and verification of software systems of all shades but is less well understood when it comes to more heterogeneous systems including physical as well as logical aspects and human actors.

The IIIf allows the formal specification of actors, policies and infrastructures within the interactive proof assistant Isabelle thus supporting automated reasoning for human centric security and privacy [9]. Existing applications of the IIIf indicate the suitability for airplane security: the IIIf has been successfully applied to Insider attacks on airplanes [11].

The IIIf includes automated analysis with attack trees and model checking. However, more intricate notions like IFC are known to be not within the reach of model checking verification because – as McLean has first realized [14] – the central property of noninterference is a property over sets of system executions rather than a property on system executions as it expresses the change of behaviours of a system by comparing various different runs of it.

In the remainder of this section, we briefly summarize the IIIf and the air-traffic monitoring and control systems we are addressing with our approach. Thereafter, we show how an air-traffic system can be specified in IIIf (Section 2) using blurring for obfuscation. Since this causes implicit information flows, we then introduce the decentralized label model as a basis (Section 3) and show subsequently how these labels are integrated into the flightradar specification (Section 4). To avoid that unauthorized users can perceive that planes on critical missions are avoiding critical

ARES 2024, July 30–August 02, 2024, Vienna, Austria

ACM ISBN 979-8-4007-1718-5/24/07
https://doi.org/10.1145/3664476.3670928

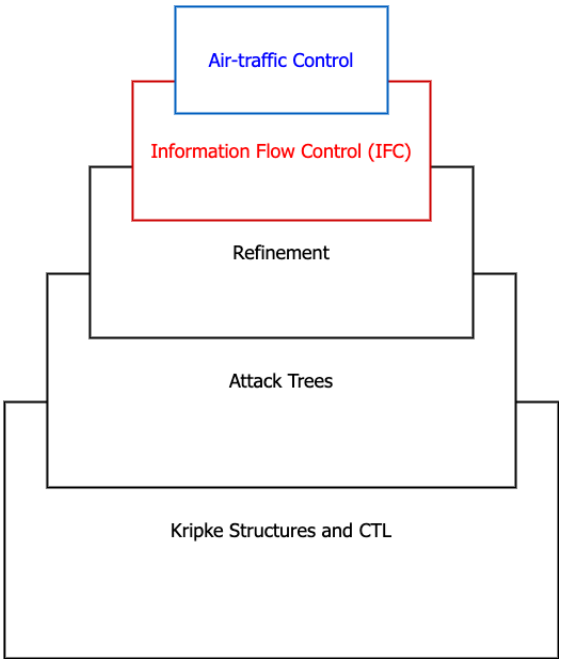

Figure 1: Structure of Isabelle Infrastructure framework (IIIf) extended with IFC for airplane monitoring.

regions – and thus giving away the criticality of plane and region – we introduce a function for hiding details of locations to all but entitled users. The idea of hiding gives rise to expressing the notion of indistinguishability for attackers and proving dynamic information flow security by noninterference (Section 5). We conclude with a summary and consideration of related work in Section 6.

1.1 Isabelle Insider and Infrastructure framework IIIf

The IIIf is an extension (a so-called object-logic) of Higher Order Logic in the interactive generic theorem prover Isabelle/HOL [17]. It thus augments the automated reasoning capabilities of Isabelle with dedicated support for modeling and proving of systems with physical and logical components, actors and policies. The IIIf has been designed for the analysis of insider threats. However, the implemented theory of temporal logic combined with Kripke structures and its generic notion of state transitions are a perfect match to be combined with attack trees into a process for formal security engineering [1] including an accompanying framework [8]. An overview of the layers of object-logics of the IIIf is provided in Figure 1: each level above is embedded into the one below; the contribution of this paper is the additional layer of Information Flow Control (IFC) that has been formalized as a generic extension. The application to Air-traffic control systems serves as an illustration but it is of general use as a template for similar applications. The source code of our extension of the IIIf with IFC and its application to flight control is submitted as a second source code submission with the paper.

A number of case studies (see Section 6) have contributed to shape the IIIf into a general framework for the state-based security analysis of infrastructures with policies and actors. Temporal logic and Kripke structures are deeply embedded into Isabelle's Higher Order logic thereby enabling meta-theoretical proofs about the foundations: for example, equivalence between attack trees and CTL statements have been established [7] providing sound foundations for applications. This foundation provides a generic notion of state transition on which attack trees and temporal logic can be used to express properties for applications.

The use of Isabelle and its Higher Order Logic for the construction of the IIIf permit a meta-theoretical approach: concepts like temporal logics, model checking, and now IFC can be formalized in the logic while at the same time these concepts can be applied to case studies.

1.2 Air-traffic information systems

The American Federal Aviation Administration (FAA) publishes radar data but systems like flightradar24 (see Figure 2) rely on planes knowing their own position quite accurately via GPS and broadcasting them over the Automatic Dependent Surveillance Broadcast (ADS-B). Partially private ADS-B spotters in various locations receive this data and forward it to the web-service. Not all planes can be spotted, for example military planes and private planes. It depends on them having ADS-B systems on board. Also, it is possible to agree with the web services not to show one's plane – or to show it in a more secure way. That is, what we propose in this paper and for which we provide the foundations.

2 AIRTRAFFIC SYSTEM IN IIIF

The IIIf allows representing infrastructures as graphs where the actors and local policies are attached to the nodes. Infrastructures

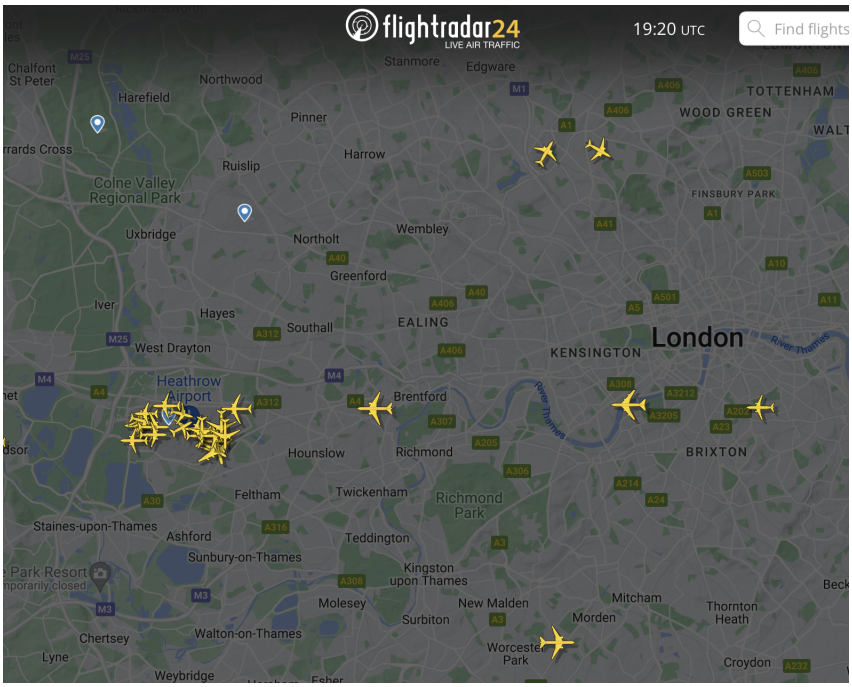

Figure 2: Flightradar systems publicly visualize airtraffic routes of airplanes, e.g. flightradar24 [4].

are the *states* of the system. We use the underlying theory of Kripke structures and temporal logics CTL in the IIIf to define a specific state transition relation for our application scenario.

The infrastructure graph is defined as a datastructure `igraph` whose components `gra`, `planes`, `routes`, and `critloc` represent the parts as: a set of pairs of locations – the coordinates of the map for the flightradar system; the identities of the airplanes at each location; the routes of each airplane and the criticality of each location. The latter is a boolean valued function (predicate) as we assume for simplicity criticality to be a boolean flag. The constructor `Lgraph` puts these components into an `igraph`.

datatype `igraph = Lgraph`
gra: `(location × location)set`
planes: `(location × location) ⇒ identity set`
routes: `identity ⇒ (location × location)list`
critloc: `(location × location) ⇒ bool`

2.1 Infrastructure state transition

The generic state transition relation uses the syntactic infix notation `I` $\rightarrow_i$ `I'` to denote that infrastructures `I` and `I'` are in this relation. To give an impression of this definition, we show here just a couple of rules that define the state transition for the action `move` because these rules will be crucial in the following.

There are two rules for the action `move`: `move` and `move_crit`. The first specifies how an airplane moves following a flight path in the non critical case where the next location on the flightpath is not critical: ¬critloc G (n, n'). It simply updates the `flight` component at the current position `(l,l')` and the next position `(n,n')` on the flightpath by removing the airplane `f` from the former and placing it on the latter position.

move:
```
G = graphI I ⟹ (l, l') ∈  gra G ⟹
f ∈ planes G (l, l') ⟹
(n,n') = hd(routes G f) ⟹ ¬ critloc G (n, n') ⟹
I' = Infrastructure
 (Lgraph
   (gra G)
   (planes G((l,l'):= planes G(l,l') - {f})
       ((n,n'):= planes G(n,n') ∪ {f}))
   (routes G(f := tl (routes G f)))
   (critloc G))
 (Localpolicy I)
⟹ I →ᵢ I'
```

However, if an airplane comes across a critical location on its flight path and if the airplane is categorized as a critical plane (facts encoded in the component `critloc` within the infrastructure graph)) then the airplane will circumvent the location. Circumvention is defined by the function `circumvent` that uses the current position `(l,l')` and the planned next one `(n, n')` on the flightpath to compute the closest alternative position.

move_crit:
```
G = graphI I ⟹ (l, l') ∈  gra G ⟹
f ∈ planes G (l, l') ⟹
(n,n') = hd(routes G f) ⟹ critloc G (n, n') ⟹
I' = Infrastructure
 (Lgraph
   (gra G)
   (planes G)((l,l'):= planes G(l,l') - {f})
              ((circumvent(l,l')(n,n')):=
                  planes G(n,n') ∪ {f}))
   (routes G(f := tl (routes G f)))
```

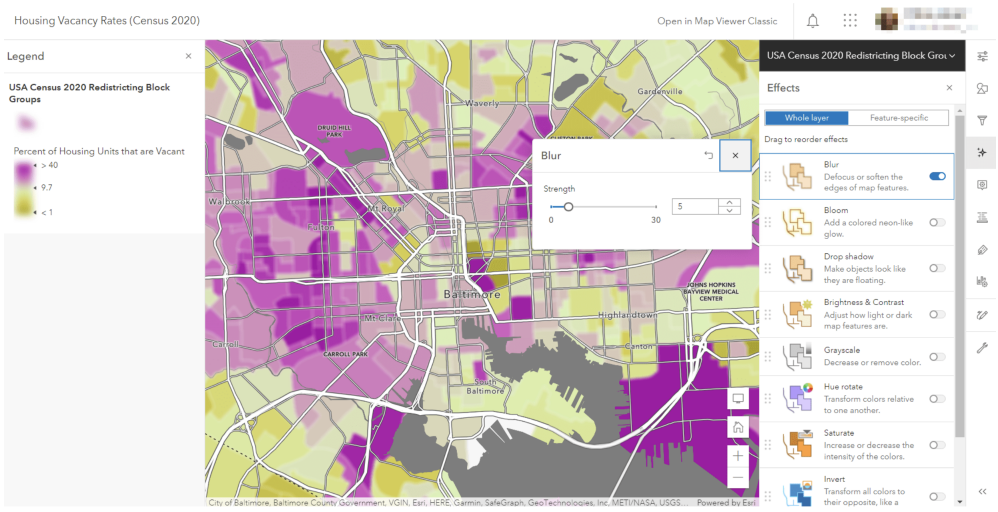

Figure 3: Geographic Information Systems (GIS) use blurring to hide private information in static maps, e.g. ArcGIS [13].

```
   (critloc G))
 (Localpolicy I)
⟹ I →ᵢ I'
```

The definition of circumvent takes into account the trajectory of the flightpath[1]. The function circumvent can be defined surprisingly easily.

```
definition circumvent
where
  circumvent(l,l')(n,n') =
     if l = n then (Suc l, n') else (n, Suc n')
```

This circumvention now would represent an illicit information flow, that is, unauthorized users learn confidential information about the criticality of the plane. To prevent this flow, we can enforce a blurring effect on the flight monitor for all unauthorized users. This idea is also used in geographic information systems (GIS) to preserve privacy: to hide some detail of maps, the maps are blurred. For example in the commercial GIS system ArcGIS [12] (see Figure 3) details of city maps can be obfuscated by blurring details. Blurring corresponds to adding "noise" to the data. Consequently, differential privacy is achieved to some extent [12]. Technically, blurring is simply achieved by a function on states that identifies locations in direct vicinity of critical locations.

```
definition vicinity
where
  vicinity (c,c') =
    {(c-1,c'),(c+1,c'),(c,c'-1),(c,c'+1),
     (c-1,c'-1),(c+1,c'+1),(c-1,c'+1),(c+1,c'-1)}
```

Blurring is then realized on infrastructure states by adapting the `planes` component that shows all airplanes at a location: for all critical positions, the planes in the vicinity are added to the critical location. That is, the points which are in vicinity of critical positions are "displayed" as being on the critical position. Thus the planes on the critical position as well as the ones all around are blurred into one position. This hides the fact that the neighbouring planes might actually be circumvented.

```
definition blur
where
 blur I = Infrastructure
  (Lgraph
    (gra (graphI I))
    (λ(l,l').if critloc(graphI I)(l,l') then
                planes(graphI I)(l,l') ∪
                planes(graphI I)`(vicinity(l,l'))
             else planes(graphI I)(l,l'))
    (graphI I))(routes (graphI I))
    (critloc (graphI I)))
  (Localpolicy I)
```

2.2 Illicit Information Flow

Analyzing the achieved information hiding, we observe that the exact position of the critical plane is not precisely visible any more. However, in terms of privacy preservation, the effect of the blurring procedure is negative: if attackers observe the flightradar system, they can observe that a plane is in fact privacy critical if it "blurs up" on a location, that is, appears to be all over the place. By observing and comparing the execution of system executions, the difference between runs of the system over different data (the same plane flying on different routes) produces differences in the observable output. Thereby, confidential, or private, values can be revealed by so-called implicit information flows: since the values of visible output variables depend on decisions made on the values of private variables, information flows implicitly – not via direct transfer from one variable to another. In GIS systems, like ArcGIS [12], this problem does not occur because they display only static maps thus attackers do not have the possibility of observing differences created by system executions on different GIS data.

[1] the points on the flightpath are consecutive which means they are direct neighbours in the plane. This is provided by an invariant of the state transition relation $\rightarrow_i$

In order to effectively address these problems we reuse the concept of Information Flow Control (IFC) designed for information flows in programming languages here for visually supported GIS systems. Before we show the IFC enhanced flightradar system, we present the formalization of the decentralized label model for the IIIf.

3 IFC FOR IIIF

In this section, we introduce the Decentralized Label Model (DLM) [15] to provide an extension to the IIIf. Labels in the original DLM are designed for program execution, program variables, variable owners and readers. That is, labels are a concept for programming languages. The concept has been successfully deployed for Java with the dedicated Jif framework [16] and more recently also for hardware description languages. Lifting this concept to infrastructures with actors and policies we simply assume the owners and readers to be actors of our systems; values are data values but can also be identities of actors as we are concerned with privacy; the variables are thus more generally state components that can hold (store) values. The definitions of labels and allowed flows can then proceed showing in Isabelle that labels form a complete lattice. This provides the basis for defining label inference, that is, how a decision can be made about illicit versus allowed information flows as well as defining the indistinguishability relation over states and the state transition relation. The actual implementation of the flow policies is then part of the definition of the infrastructure in the following section.

3.1 Labels

Every value (data or identity) that is used or communicated within the evolution of the infrastructure, that is, in the execution of the state transition has a label associated to it representing the *owner* of that item. Additionally, a set of *readers* – the actors (principals [15]) allowed to observe that value – are assigned to it. Labels in the DLM are now sets of allowed *flows* (pairs of actors) as there may be a range of permitted flows for a variable (a state component holding values). Therefore, labels accumulate all possible flows. Thus, a label is a set of *label components* summarizing the allowed flows for a single owner o. A label component is a pair (o, R) specifying the flows (o, r) for $r \in R$ that is the owner o permits all readers to observe. A label is a set of components. In order to guarantee that this definition yields a lattice, we complement labels by allowing all flows for actors that have no components: permitted flows of a label are the union of all flows of all components of a label L unified with the set of all flows (o, r) for every o for which there is *no* component (o, R) for some R in L. For those components o, the label set L is additionally complemented by adding all flows (o, r) for every principal r. For example, for the label

$$L_{ex} \equiv \{\text{Alice} : \{\text{Eve}\}; \text{Bob} : \{\text{Alice}\}\}$$

we have

$$[[L_{ex}]] \equiv \{(\text{Alice}, \text{Alice}), (\text{Alice}, \text{Eve}), (\text{Bob}, \text{Bob}), (\text{Bob}, \text{Alice}), (\text{Eve}, \text{Eve}), (\text{Eve}, \text{Alice}), (\text{Eve}, \text{Bob})\}.$$

assuming that Alice, Bob, and Eve are all the actors in the infrastructure.

3.2 Label lattice

With this definition of labels as sets of allowed flows, the set of labels are a complete lattice if combined with the following partial order relation on labels defined via their flow sets.

$$L_0 \sqsubseteq L_1 \equiv [[L_0]] \supseteq [[L_1]]$$

Upper and lower bounds exist naturally as we are using the subset relation on flows.

$$\begin{aligned} L_0 \sqcup L_1 &\equiv [[L_0]] \cap [[L_1]] \\ L_0 \sqcap L_1 &\equiv [[L_0]] \cup [[L_1]] \end{aligned}$$

A value (and thus information) may flow from a variable (state component) labeled L_0 to one labeled L_1. This flow is permitted only if the labels are ordered. That is, the labels must be equal or L_1 is more restrictive than L_0: $L_0 \sqsubseteq L_1$.

It is useful to show that labels with the ordering are a lattice because this allows using general lattice operators like upper (join) and lower bounds (meet) in order to infer labels of compound expressions. Since lattices are part of Isabelle's theory library, we can simply reuse these operators and their properties for the formalization of the DLM in IIIf.

$$\begin{aligned} \mathit{owners}(L_1 \sqcup L_2) &\equiv \mathit{owners}(L_1) \cup \mathit{owners}(L_2) \\ \mathit{readers}(L_1 \sqcup L_2, O) &\equiv \mathit{readers}(L_1, O) \cap \mathit{readers}(L_2, O) \end{aligned}$$

Corresponding equations for the operation meet ($\sqcap$) are defined dually.

In the application, we assume initially that all infrastructure state components and the state transition relation are labeled consistently with the policy. This means, that the state transition only allows information flows if they flow "up" with respect to the lattice order defined above. To verify this in practice, we need to define secure semantics rules of the state transition relation. This means that they must respect the labels assigned to the state components and admit flows only if they are consistent with the policy.

Moreover, we want to additionally verify that there are no *implicit flows* as has been identified in the blur example in Section 2.2. These are information flows where values of low classified variable may depend on higher classified control variables thereby representing illicit leaks of information. To address this, we define indistinguishability as a relation over the state transition (Section 5). As a motivation, we first introduce DLM labels for IFC into the airtraffic system.

4 IFC ENHANCED AIRTRAFFIC SYSTEM

For the extension with the IFC, each component of the infrastructure graph datatype `igraph` is paired with a DLM label of type `dlm` as first element. In addition, the (secret) circumvented position of a plane is added as a new component `critpos`. This aims at avoiding the negative effect of blur that reveals information by implicit flows and uses data hiding instead. The updated `igraph` datatype looks now as follows.

```
datatype igraph = Lgraph
gra: dlm × (location × location)set
planes: dlm × ((location × location) ⇒ identity set)
routes: dlm × (identity ⇒ (location × location)list)
critloc: dlm × ((location × location) ⇒ bool)
critpos: dlm × (identity ⇒ (location × location)option)
```

4.1 Infrastructure state transition

As previously, there are two rules for the action `move`: `move` and `move_crit`. The first rule is again for the case ¬critloc G (n, n') the next position on the flightpath is not critical. Here, the rule again simply updates the `planes` component at the current position `(l,l')` and the next position `(n,n')` on the flightpath by removing the airplane `f` from the former and placing it on the latter position. Additionally now, the action also sets back the `critpos G` flag for the flight to delete a previous "circumvented" position in case the previous position had been critical (see the following action `move_crit` for more detail).

move:

```
G = graphI I ⟹ (l, l') ∈  gra G ⟹
f ∈ planes G (l, l') ⟹
(n,n') = hd(routes G f) ⟹ ¬ critloc G (n, n') ⟹
I' = Infrastructure
 (Lgraph
   (gra G)
   (fst (planes G),
   ((snd(planes G))((l,l'):=(snd(planes G))(l,l')-{f})
       ((n,n'):= (snd(planes G))(n,n') ∪ {f}))
   (fst (routes G),
   (snd (routes G))(f := tl (snd(routes G) f)))
   (critloc G)
   (fst (critpos G), (snd (critpos G))(f := None))))
 (Localpolicy I)
⟹ I →ᵢ I'
```

The case for moving over a critical location is different from the previously defined model in Section 2. The plane is not circumvented in the component `planes` but apparently remains "on course". Instead, the new additional component registers that the flight is in fact circumvented. Modeling the circumvention this way allows to hide the actual position easily by using the planes component as the "public" (or official) position while hiding the real position in the `critpos` position.

move_crit:

```
G = graphI I ⟹ (l, l') ∈  gra G ⟹
f ∈ planes G (l, l') ⟹
(n,n') = hd(routes G f) ⟹ critloc G (n, n') ⟹
I' = Infrastructure
 (Lgraph
   (gra G)
   (fst(planes G),
   (snd(planes G))((l,l'):= (snd(planes G))(l,l')-{f})
              ((n,n'):= (snd(planes G))(n,n')∪{f}))
   (fst(routes G),
   (snd(routes G)(f := tl (snd(routes G) f)))
   (critloc G)
   (fst (critpos G),
   (snd(critpos G))(f → (circumvent (l,l')(n,n'))))))
 (Localpolicy I)
⟹ I →ᵢ I'
```

The definition of circumvent is the same as previously defined.

```
definition circumvent
where
  circumvent(l,l')(n,n') =
     if l = n then (Suc l, n') else (n, Suc n')
```

That is, the planes that are circumventing critical positions are registered in the `planes` component of the infrastructure graph as being *on* the critical position thus hiding the fact that these planes are actually circumvented. The actual position is, however, not lost as it is being stored in the new `critpos` component of the infrastructure graph.

Similar to the idea of having a blur function, we can now define a function hide that obfuscates information contained in an infrastructure graph from observers whose security clearance is below a certain threshold.

```
definition hide
where
 hide (Lgraph g pl r cl cp) a =
  let G = (Lgraph g pl r cl cp) in
  (Lgraph
   (if (fst g ⊑ a) then g else (fst g, {}))
   (if (fst pl ⊑ a) then pl else (fst pl, (λ x. {})))
   (if (fst r ⊑ a) then r else (fst r, (λ x. [])))
   (if (fst cl ⊑ a) then cl else (fst cl,  (λ x. True)))
   (if (fst cp ⊑ a) then cp else (fst cp, (λ x. None))))
```

The above function hide evaluates the `dlm` labels on each component of an infrastructure graph comparing them to the observers level `a`. Observers have access to the components values, if their clearance label is above the components access label in the `dlm` ordering. Otherwise, a sanitized dummy value is output. For example, hide obfuscates the fields `critloc` and `critpos` that contain security and privacy critical information (like the real circumvented position) by replacing them with the trivial predicate `(λx.True)` returning True for all inputs for the former and the everywhere undefined function `(λx.None)` for the latter.

We are going to define next a general notion of indistinguishability which conceptualizes the hiding effect of the function hide. Different to static notions of confidentiality used for judging data privacy, like differential privacy, indistinguishability is an equivalence relation on states that is used to express security with respect to dynamic information flows in processing systems. More precisely, the notion of noninterference shows security as absence of illicit information flows in dynamic system evolutions. We prove noninterference for the airtraffic control system.

5 NONINTERFERENCE

IFC can be enforced on programming systems by a concept called Noninterference (NI) [5]. NI is a an equivalence property over sequences of state transitions (or sequences of events – depending on the system view). Intuitively, it characterizes that from a certain viewpoint various executions of a system do not reveal any secret information. This viewpoint allows to integrate NI with the idea of security classes that have different observation (or alteration) rights on information. This viewpoint, often identified by access control classes, categories or compartments of user groups, gives rise to a notion of *indistinguishability*: from the viewpoint of an observer, that is, a specific access control class. the system states appear equal. Given this concept of indistinguishability of states of a system, the notion of noninterference states that if any two states of a system are indistinguishable, then so are future next states of those. Noninterference thus lifts the notion of indistinguishability of states to those of execution sequences of states.

More formally, we define the indistinguishability relation over infrastructure graphs by a case statement that analyses each of the graph's components.

```
definition indistinguishability_g
where indistinguishability_g g₀ a g₁ ≡
 ((if (fst (gra g₀)  ⊑ a)
   then (snd(gra g₀) = snd(gra g₁)) else True) ∧
  (if (fst (planes g₀) ⊑ a)
   then (snd(planes g₀) = snd(planes g₁)) else True) ∧
  (if (fst (routes g₀) ⊑ a)
   then (snd(routes g₀) = snd(routes g₁)) else True) ∧
  (if (fst (critloc g₀) ⊑ a)
   then (snd(critloc g₀) = snd(critloc g₁)) else True) ∧
  (if (fst (critpos g₀) ⊑ a)
   then (snd(critpos g₀) = snd(critpos g₁)) else True))
```

Indistinguishability thus requests that all components of the graphs s_0, s_1 that are visible from the observer `a`'s viewpoint must be equal. Otherwise – if the component is not visible – there are no constraints which is expressed by `True` for each component.

We then lift the indistinguishability relation over infrastructure graphs to whole infrastructures allowing us to reason about indistinguishability of system states s_0 and s_1 as $s_0 \sim_a s_1$. Isabelle allows us to define behind the name of the indistinguishability relation, the mathematical notation $\sim_a$ commonly used for indistinguishability.

```
definition indistinguishability ("(_ ~(_) _)")
where s₀ ~a s₁ =
      (indistinguishability_g (graphI s0) a (graphI s1))
```

Information must only flow from objects to subjects if the objects level is below the subjects level. This conforms to the policy that information may only flow up in the security class lattice ordering of the `dlm`. Indistinguishability formalizes for a system state that information contained in a state is visible to higher classified subjects only. Indistinguishability statically expresses the absence of illicit information flows.

Based on the indistinguishability relation we can also formalize what it means that the visibility of information is not compromised during the execution of the state transition relation. This means that information does not flow across the set boundaries not even implicitly during system execution. This formal property of the dynamics of indistinguishability is coined *noninterference*: If two states s_0, s_1 are indistinguishable, that is, $s_0 \sim s_1$, then for any next state s_0' of s_0, that is $s_0 \rightarrow s_0'$, there exists a state s_1' reachable from s_1, that is, with $s_1 \rightarrow^* s_1'$, such that $s_0' \sim s_1'$. The states s_0' and s_1' are also indistinguishable and thus indistinguishability is preserved by the state transition relation.

Given the definition of indistinguishability, we can prove NI as a theorem. Apart from being a theoretical exercise, the process of proving this property is a practical tool because it exhibits a range of preconditions necessary for IFC of the flightradar application:

- the states s_0 and s_1 have to be reachable by some (initial) configuration I, that is, $I \rightarrow^* s_0$ and $I \rightarrow^* s_1$;
- the location map of the graph, the routes, and the current locations of the planes need to have initially the same security classifications;
- the security level of the critical position needs to be in a class above the other components.

These conditions have emerged when trying to prove the noninterference property. They reflect the dependencies of the control flows of the variables. Since we are assuming reachable states, these initial conditions hold for all reachable states – a fact that we established by proving corresponding invariants as a prerequisite for the proof of the following theorem.

```
theorem noninterference:
  I →* s₀ ⟹ I →* s₁ ⟹
  fst (gra (graphI I)) = fst (planes (graphI I)) ⟹
  fst (planes (graphI I)) = fst (routes (graphI I)) ⟹
  fst (routes (graphI I)) = fst (critloc (graphI I)) ⟹
  fst (critloc(graphI I)) ⊑ fst (critpos (graphI I) ⟹
  s₀ ~a s₁ ⟹ s₀ → s₀' ⟹
  ∃ s₁'. s₁ →* s₁' ∧ s₀' ~a s₁'
```

Given an initial setting of the security classes, we can prove in the IIIf that indistinguishability is preserved by the state transition relation. Therefore, attackers with security clearance below `a` cannot learn anything about critical values contained in the fields `critloc` and `critpos` – even if they have access to lower components. From their perspective there are no visible changes because there are no implicit information flows down the security hierarchy. Thus there are no leaks from the privacy critical values to unauthorized users.

6 CONCLUSIONS AND RELATED WORK

In this paper, we have presented the extension of the IIIf framework by IFC motivating and illustrating it by the application to flight monitoring systems. We have formalized a flight radar system including a blurring function that permits to blur the exact position of a plane in case the flight passes over a critical region. We have then argued that although this blurring somehow hides the real position it gives away that it is a critical flight. To address this issue we propose to control information flows. Actors in the security class of external users cannot learn anything about critical missions visible only to actors in the higher security class of, for example, air traffic controllers. We have motivated the concept by a hiding function similar to the previous blur-function. We then introduced a notion of indistinguishability that allows to precisely characterize the visibility of objects in states from any given vantage point. Indistinguishability permits to prove the noninterference property showing the absence of implicit information flows during the execution of a system. The process of proving is not only a proof of concept for the theoretical concept but serves as a tool to exhibit the initial security classification of the state components as well as invariant conditions.

With respect to relevant related work on noninterference for airplane security, there appears to be very little. A notable exception is Hill and Lake's work [6] on applying noninterference analysis to code used in avionics systems. In comparison to our work this previous work uses noninterference for analysis of software within an airplane as part of the technical control system different from our high level view of protection of security and privacy related data within airtraffic control and in relation to geographic information systems (GIS). From that perspective, clearly, the application of the IIIf to airplane-security policies [10, 11] is most closely related. However, these works are focused on the security policies within the airplane addressing the verification of airplane security against insider attacks by applying model checking within the IIIf. This

work verifies, for example, the two-person rule that has been implemented after a tragic insider attack in 2015 to ensure that always two crew members are in the cockpit at any time. Such security properties of policies are similar to safety properties and can very well be expressed as temporal properties of state transition paths (for example using temporal logics like CTL). However, more subtle security properties related to implicit information flows require more complex notions like noninterference to be formalized and verified. Exceeding the previous work we have realized this in the current paper. Besides being more accurate, indistinguishability and noninterference add a dynamical security aspect. Blurring works fine for static data, like maps, but when this data is animated information leaks appear. A solution to this problem we have presented in this paper.

REFERENCES

[1] CHIST-ERA. 2016. SUCCESS: SecUre aCCESSibility for the internet of things. http://www.chistera.eu/projects/success.

[2] Dorothy Denning. 1982. *Cryptography and Data Security*. Addison-Wesley. reprinted with corrections, Janauary 1983.

[3] D. E. Denning and P. J. Denning. 1977. Certification of programs for secure information flow. *Commun. ACM* 20, 7 (1977).

[4] Flightradar24. 2024. Live Air Traffic. https://www.flightradar24.com/ Accessed 24. 1. 2024.

[5] J. Goguen and J. Meseguer. 1982. Security Policies and Security Models. In *Symposium on Security and Privacy, SOSP'82*. IEEE Computer Society Press, 11–22.

[6] M.G. Hill and T.W. Lake. 2000. Non-interference analysis for mixed criticality code in avionics systems. In *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*. 257–260. https://doi.org/10.1109/ASE.2000.873672

[7] F. Kammüller. 2018. Attack Trees in Isabelle. In *20th International Conference on Information and Communications Security, ICICS2018 (LNCS, Vol. 11149)*. Springer.

[8] F. Kammüller. 2019. Combining Secure System Design with Risk Assessment for IoT Healthcare Systems. In *Workshop on Security, Privacy, and Trust in the IoT, SPTIoT'19, colocated with IEEE PerCom*. IEEE.

[9] F. Kammüller. 2022. Isabelle Insider and Infrastructure framework with Kripke strutures, CTL, attack trees, security refinement, and example Corona Warning App. Available at https://github.com/flokam/CoronaApp..

[10] Florian Kammüller and Manfred Kerber. 2016. Investigating Airplane Safety and Security against Insider Threats Using Logical Modeling. In *IEEE Security and Privacy Workshops, Workshop on Research in Insider Threats, WRIT'16*. IEEE.

[11] Florian Kammüller and Manfred Kerber. 2021. Applying the Isabelle Insider framework to airplane security. *Science of Computer Programming* 206 (2021). https://doi.org/10.1016/j.scico.2021.102623

[12] Diana Lavery. 2022. The perfect layer effect for differentially-private data: Blur. https://www.esri.com/arcgis-blog/products/arcgis-online/mapping/blur-differential-privacy/ accessed 2.12.2023.

[13] Diana Lavery. 2022. Visual Example of Blur in ArcGIS. https://www.esri.com/arcgis-blog/wp-content/uploads/2022/05/BG_blur.png accessed 12.05.2024.

[14] John Mclean. 1994. A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions. In *In Proc. IEEE Symposium on Security and Privacy*. 79–93.

[15] A. C. Myers and B. Liskov. 1999. Complete, Safe Information Flow with Decentralized Labels. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE.

[16] Andrew C. Myers, Lantian Zheng, Steve Zdancewic, Stephen Chong, and Nathaniel Nystrom. 2006. *Jif 3.0: Java information flow*. http://www.cs.cornell.edu/jif

[17] T. Nipkow, L. C. Paulson, and M. Wenzel. 2002. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. LNCS, Vol. 2283. Springer-Verlag.