Checking and Enforcing Security Through Opacity in Healthcare Applications

Rym Zrelli[1(✉)], Moez Yeddes[2], and Nejib Ben Hadj-Alouane[1]

[1] OASIS Reasearch Lab (ENIT), University of Tunis El Manar, Tunis, Tunisia
rym.zrelli@gmail.com
[2] OASIS Reasearch Lab (INSAT), University of Carthage, Tunis, Tunisia

Abstract. The Internet of Things (IoT) is a paradigm that can tremendously revolutionize health care thus benefiting both hospitals, doctors and patients. In this context, protecting the IoT in health care against interference, including service attacks and malwares, is challenging. Opacity is a confidentiality property capturing a system's ability to keep a subset of its behavior hidden from passive observers. In this work, we seek to introduce an IoT-based heart attack detection system, that could be life-saving for patients without risking their need for privacy through the verification and enforcement of opacity. Our main contributions are the use of a tool to verify opacity in three of its forms, so as to detect privacy leaks in our system. Furthermore, we develop an efficient, Symbolic Observation Graph (SOG)-based algorithm for enforcing opacity.

1 Introduction

Real-world usage of IoT in health-care necessitates the dealing with new security challenges. In fact, and since this type of application would handle medical and personal information, their employment carries serious risks for personal privacy. Accordingly, it is paramount to protect any sensitive data against deduction by third-parties to avoid the compromise of privacy. The most common security preservation practice is the use of cryptographic techniques. However, these techniques do not provide perfect security as the inference of critical information from non-critical ones remains a possibility. The discovery of vulnerabilities of simple crypto-systems like that of the Needham-Schroeder public key protocol [10] proved that cryptography is not enough to guarantee the privacy of information. Furthermore, the various techniques available are computationally intensive. This is why they cannot be immediately adopted in IoT where the network nodes are powered by battery. To facilitate the adoption of IoT in health-care, we need formal (preferably automated) verification of security properties. Formal verification ensure that the system's design conforms to the desired behavior. Information flow properties are the most formal security properties. In fact, various ones have been defined in the literature including non-interference, intransitive non-interference and others (e.g. secrecy, and anonymity). Interested in

L. Braubach et al. (Eds.): ICSOC Workshops 2017, LNCS 10797, pp. 161–173, 2018.
https://doi.org/10.1007/978-3-319-91764-1_13

confidentiality properties, we consider opacity, a general information flow property, to analyze IoT privacy in a heart attack detection system. Opacity's main interest is to formulate the need to hide information from a passive observer. It was first introduced in [12] and was later generalized to transition systems [4]. It has since, been studied several times allowing the formal verification of system models. Its wide study led to the birth of several variants as well as verification and enforcement techniques. If classified according to the security policy, then we are dealing with simple, K-step, initial, infinite as well as strong and weak opacity alongside their extensions (e.g., K-step weak and K-step strong opacity). The efforts of these studies also made possible not only opacity verification, but also its assurance via supervision [5], [14] or enforcement [7]. Several IoT-based solutions [1,8] for healthcare are known in the literature to deal with privacy issues. A key limitation of these studies is that they have been using cryptographic methods.

In this paper, we wish to show the practical use of our SOG-based approach and the relevance of the use of opacity in the real world through the synthesis of an opaque IoT-based heart attack detection system. Building on the SOG-based verification approach developed in [3], the purpose is to verify opacity in three of its forms (simple, K-step weak opacity and K-step strong opacity) to detect security violations in our synthesized system. Then to contribute an algorithmic approach that enforces simple opacity by padding the system with minimal dummy behavior.

This paper is organized as follows: Sect. 2 establishes all necessary basic notions including the SOG structure and the opacity property. In Sect. 3, we detail the case study. In Sect. 4, we illustrate the practical usefulness of the opacity verification approach in the heart attack detection system. Section 5 details our proposed approach to enforce simple opacity. Finally, we conclude in Sect. 6, and list some potential future works.

2 Preliminaries

2.1 Petri Nets, WF-net and oWF-nets

To model the services under consideration in our case study, we use Petri nets. A service can be considered as a control structure describing its behavior in order to reach a final state. We can represent it using a Workflow net, a subclass of Petri nets. A WF-net satisfies two requirements: it has one input place i and one output place o, and every transition t or place p should be located on a path from place i to place o. To model the communication aspect of a service, we can use open Work-Flow nets which is enriched with communication places representing the (asynchronous) interface. Each communication place represents a channel to send or receive messages to or from another oWF-net.

Definition 1 (oWF-net [11])
An open Work-Flow net is defined by a tuple $\mathcal{N} = (P, T, F, W, m_0, I, O, m_f)$*:*

- (P, T, F, W) *is a WF-net;*
  - P *is a finite set of places and* T *a finite set of transitions;*
  - F *is a flow relation* $F \subseteq (P \times T) \cup (T \times P)$;
  - $W : F \to \mathbb{N}$ *is a mapping allocating a weight to each arc.*
- m_0 *is the initial marking;*
- I *is a set of input places and* O *is a set of output places* ($I \cup O$*: the set of interface places).*
- m_f *is a final marking.*

Having the same semantics as Petri nets, the behavior of WF-nets and oWF-nets can be represented by Labeled Transition Systems (LTS).

2.2 Labeled Transition System

An LTS is defined as follows:

Definition 2 (Labeled Transition System)
A Labeled Transition System is a 4*-tuple* $\mathcal{G} = (Q, q_{init}, \Sigma, \delta)$*:*

- Q*: a finite set of states;*
- q_{init}*: the initial state;*
- Σ*: actions' alphabet;*
- $\delta : Q \times \Sigma \to Q$*: the transition function where:* $q,\ q' \in Q$ *and* $\sigma \in \Sigma$, $\delta(q, \sigma) = q'$ *meaning that an event* σ *can be executed at state* q *leading to state* q'*.*

The language of an LTS $\mathcal{G}$ is defined by $L(\mathcal{G}) = \{t \in \Sigma^*, q_0 \xrightarrow{t} q_f\}$. An LTS can be considered as an automaton where all states are accepting final states.

To reflect the observable behavior of an LTS, we specify a subset of events $\Sigma_o \subseteq \Sigma$ and $\Sigma - \Sigma_o = \Sigma_u$ where Σ_o is the set of events visible to a given observer and Σ_u is the set of events which are invisible to said-observer. The behavior visible is defined by the projection P_{Σ_o} from Σ^* to Σ_o^* that removes from a sequence in Σ^* all events not in Σ_o. Formally, P_o: $\Sigma^* \to \Sigma_o^*$ is defined:

$$\begin{cases} P_{\Sigma_o}(\epsilon) = \epsilon; \\ P_{\Sigma_o}(u \cdot \sigma) = \begin{cases} P_{\Sigma_o}(u) \text{ if } \sigma \notin \Sigma_o; \\ P_{\Sigma_o}(u) \cdot \sigma \text{ otherwise.} \end{cases} \end{cases} \quad \text{Where: } \sigma \in \Sigma \text{ and } u \in \Sigma^*.$$

2.3 Opacity

Opacity's main interest is in capturing the possibility of using observations and prior-knowledge of a system's structure to infer secret information. It reflects a wide range of security properties. Opacity's parameters are a secret predicate, given as a subset of sets or traces of the system's model, and an observation function. This latter captures an intruder's abilities to collect information about the system. A system is, thus, opaque w.r.t. the secret and the observation function, if and only if for every run that belongs to the secret, there exists another run with a similar projection from the observer's point of view and that does not belong to the secret [5,6,9]. In this paper, we focus on 3 opacity variants as defined in [6]: simple, K-step weak and K-step strong opacity.

Definition 3 (Simple opacity [6])
Given an LTS $\mathcal{G} = (Q, q_0, \Sigma, \delta)$ *with* $\Sigma_o \subseteq \Sigma$ *is the set of observable events and* $S \subseteq Q$ *is the set of secret states. The secret* $S \subseteq Q$ *is opaque under the projection map* P_{Σ_o} *ou* (G, P_{Σ_o}) *– opaque iff:* $\forall u \in L_S(G), \exists v \in L(\mathcal{G}) : (v \approx_{\Sigma_o} u) \wedge (v \notin L_S(G))$.

While simple opacity deals with the non-discloser of the fact that the system is currently in a secret state, K-step weak opacity ensures that the system wasn't in a secret state K observable events ago, and K-step strong opacity formulates the need to make sure that, K-steps backwards, the system does not end, and have not crossed any secret states.

2.4 Symbolic Observation Graph

The SOG is an abstraction of the reachability graph. It is constructed by exploring a system's observable actions which are used to label its edges. The unobservable actions are hidden within the SOG nodes named aggregates. The definition of an aggregate and that of the SOG are given in the following:

Definition 4 (Aggregate)
Given an LTS $\mathcal{G} = (Q, q_0, \Sigma, \rightarrow, \delta)$ *with* $\Sigma = \Sigma_o \cup \Sigma_u$. *An aggregate* a *is a non empty set of states satisfying:* $q \in a \Leftrightarrow Saturate(q) \subseteq a$ *where:* $Saturate(q) = \{q' \in Q : q \xrightarrow{w} q' \text{ and } w \in \Sigma_u^*\}$.

Definition 5 (Deterministic SOG)
A deterministic $SOG(\mathcal{A})$ *associated with an LTS* $\mathcal{G} = (Q, q_0, \Sigma_o \cup \Sigma_u, \delta)$ *is an LTS* $(A, a_0, \Sigma_o, \Delta)$ *where:*

1. A *a finite set of aggregates with:*
   (a) $a_0 \in A$ *is the initial aggregate s.t.* $a_0 = Saturate(q_0)$;
   (b) *For each* $a \in A$, *and for each* $\sigma \in \Sigma_o$, $\exists q \in a, q' \in Q$: $q \xrightarrow{\sigma} q' \Leftrightarrow \exists a' \in A : a' = Saturate(\{q' \in Q, \exists q \in a \text{ with } q \xrightarrow{\sigma} q'\}) \wedge (a, \sigma, a') \in \Delta$;
2. $\Delta \subseteq A \times \Sigma_o \times A$ *is the transition relation.*

3 Motivating Scenario

Heart disease is the first cause of morbidity and mortality in the world, accounting for 28.30% of total deaths each year in Tunisia alone [13]. Investment in preventive health care such as the use of IoT monitoring devices may help lower the cost of processing and the development of serious health problems. Integrating clinical decisions with electronic medical records could decrease medical errors, reduce undesirable variations in practice, and improve patient outcomes.

Our case study considers IoT integration with cloud computing. We use a connected bracelet, fog nodes, a private and a public Cloud, and a mobile application, which together form a medical application. This latter provides continuous monitoring of the vital data of a given patient. Regular or routine measurements could help to detect the first symptoms of heart malfunction, and makes it possible to immediately trigger an alert. The vital information collected by the

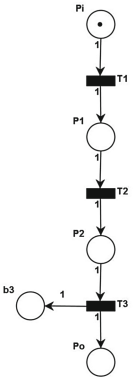
bracelet includes cardiac activity, blood pressure, oxygen levels and, temperature. As mentioned earlier, we consider an IoT application in a hybrid cloud/fog environment. The cloud [16] is considered as a highly promising approach to deliver services to users, and provide applications with low-cost elastic resources.

Public clouds provide cheap scalable resources. Making it useful for analyzing the patient's data which would be costly as it requires extensive computing and storage resources. However, we must take into account that storage of health records on a public environment is a privacy risk. To avoid such security leaks, we could deploy the application on a secure private cloud. But seeing this latter's limited resources, this may degrade the overall performance. To prevent this, the workflow can be partitioned between a private cloud and a public one. Therefore, the confidential medical data will be processed on the private cloud. Other workflow actions can be deployed on the public cloud dealing with anonymized data. The use of a cloud-based framework poses the problem of delay when sending and receiving data between the objects and geographically far cloud resources thus jeopardizing the patients' well-being given that triggering timely responses is the purpose of this data. To resolve this issue, data gathering can be moved from the cloud domain to that of the fog [2]. Bringing this action closer to the connected object shortens the transmission time, and reduces the amount of data transferred to the cloud. The proposed workflow is described as follows:

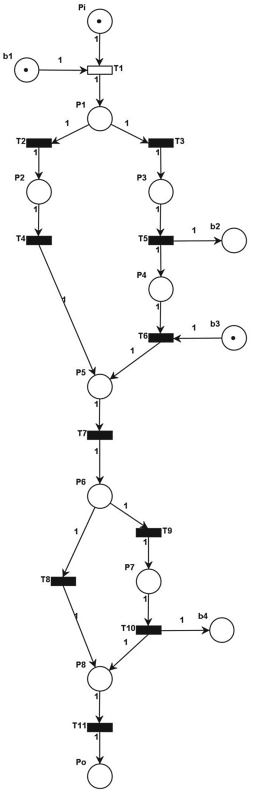
- A patient may register via the mobile app by entering his information. This information include personal data and medical history (personal and family medical histories, surgical history, drug prescriptions, and the doctors' notes).
- The patient's medical history is then transmitted to the private cloud. After reception, this latter anonymizes the data by stripping off all that could identify the patient leaving only medical data, which it sends to the public cloud.
- The public cloud receives the anonymized data, and proceeds to the classification attaching to each medical file a class.
- The patient is equipped with a measuring bracelet connected to the processing components (Fog nodes). The data sent to the fog domain is a set of vital data recorded over a period of time.
- The fog node collects the data then compares it to its predecessors, searching for any vital signs changes. When the node determines that a change has occurred, it sends the data to the private cloud.
- The private cloud links the gathered data with the patient, transmitting this data and the class ascribed to the patient, to the public cloud.
- The public cloud reads the data, analyzes it, and then provides results. When the risk of heart attack is detected, it immediately notifies the patient's app.

4 Modeling and Verification

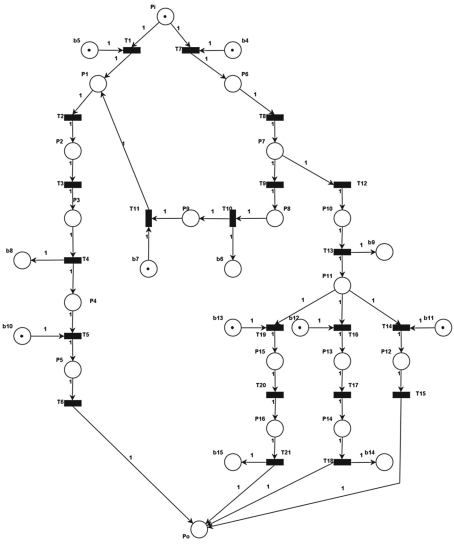
The case study contains five services, namely, a connected bracelet (Br), a fog node (Fog), a private cloud (CPr), a public cloud (CPub), and a smartphone application (App). Figure 1 depicts the oWF-nets of the Br, Fog, CPr, CPub and the App, respectively. We note that the transitions entailing the sending



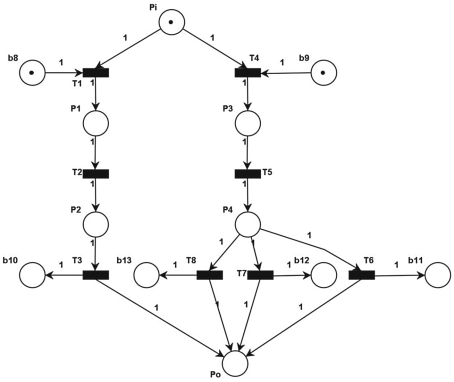
(a) Bracelet oWF-net.



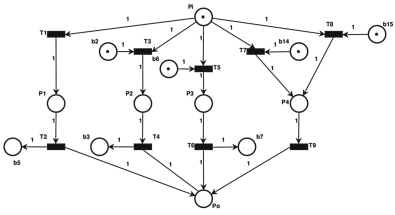
(b) Fog oWF-net.



(c) Private cloud oWF-net.



(d) Public cloud oWF-net.



(e) Application oWF-net.

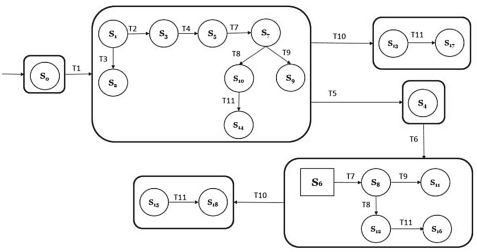
Fig. 1. Case study oWF-nets.

(respectively reception) of a messages are indicated by adding a ! (respectively a ?) mark. In this case study, we want to illustrate the ability of the SOG-based verification approach to meet privacy demands. The first step is to create the underlying LTS of each oWf-net. Secondly, we identify the observable and unobservable actions of each net as well as the secret states. Then, we build the SOG models from each net's LTS verifying, at the same time, their opacity.

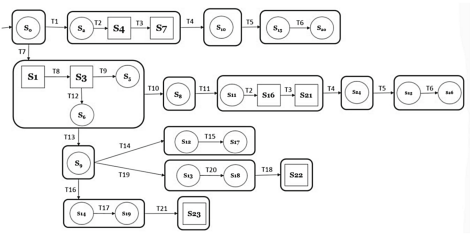
The Br workflow (Fig. 1(a)) starts by collecting data (T_1), which will then be sent to the closest Fog node. Next it creates the message comprising the data (T_2) and sends this message (T_3?). Not having any security requirements for the bracelet, thus, there is no need to check its opacity.

The Fog WS (Fig. 1(b)) has an internal set of operations, and a set of external cooperative ones. After receiving the data (T_1!), we consider two scenarios. The first is when the Fog communicates for the 1st time with the bracelet (T_3). In this case, it sends a request (T_5?) to the App to retrieve data from the patient's medical history. Then, it will receive these data through (T_6!). The second scenario begins by selecting the last recorded data (T_4). The next step is to compare (T_7) the data retrieved by one of the mentioned scenarios with the data sent by the Br. When the node detects a change in values (T_9), it will immediately transmit the data to CPr (T_{10}?). If there is no change (T_8), the Fog doesn't perform any processing. Finally, the new data will be stored locally in the Fog (T_{11}). To ensure the privacy of fog secret information, we define the secret state $S = \{S_6\}$ which is related to receiving patient's medical history. To conform with the security needs, the observable transitions of the Fog are $\Sigma_o = \{T_1!, T_5?, T_6!, T_{10}?\}$, while the unobservable part is $\Sigma_u = \{T_2, T_3, T_4, T_7, T_8, T_9, T_{11}\}$. Using this data, we proceed to the opacity verification which is done while creating the SOG-abstraction of the model. We get the SOG in Fig. 2(a) and we can conclude that the fog's SOG is both simple, and K-step weakly and strongly opaque.

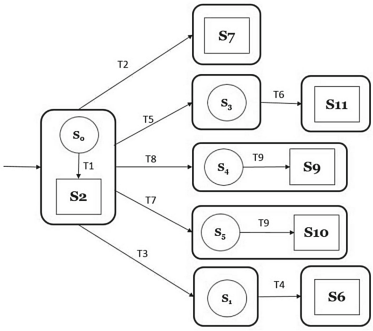
The CPr workflow (Fig. 1(c)) contains two scenarios. The first one starts by receiving the data of a registered patient (T_1!). The CPr subsequently proceeds with the recording (T_2) and the anonymization (T_3) of the received data. The anonymised data will then be transmitted to the CPub (T_4?). After receiving (T_5!) the class, this latter is associated with the patient (T_6). The second scenario starts when the CPr receives (T_7!) the data sent by the Fog. The CPr combines the data with the patient by searching for its ID (T_8). If the ID cannot be found (T_9), the CPr sends a request to the App so that the patient re-enter his information (T_{10}?). Thereafter, it receives the requested data (T_{11}!) and it pursues the first scenario. For the second case, when the ID is found, the CPr transmits the data and the class to which the patient belongs to the CPub (T_{13}?). Afterwards, the CPr receives and records respectively 3 types of messages, each one belongs to an alert type: low (T_{14}! & T15), medium (T_{16}! & T_{17}) and high (T_{19}! & T_{20}). To protect the privacy of patients, the CPr need to hide the update procedure performed on the patient's personal information. It must keep secret the states related to the patient registration (S_4 & S_{16}) and the anonymization of his data (S_7 & S_{21}). It is also required to withhold secret the states related to sending alerts (S_{22} & S_{23}). So the set of



(a) The SOG of the Fog Node.



(b) The SOG of the Private Cloud.



(c) The SOG of the Mobile Application.

Fig. 2. The SOGs of the case study WSs.

secret states for the CPr is $S = \{S_1, S_3, S_4, S_7, S_16, S_{21}, S_{22}, S_{23}\}$, where S_1 stands for the marking related to the reception of the data sent by the fog, while S_3 reflects that related to patient ID search. The observable transitions of the CPr are $\Sigma_o = \{T_1!, T_4?, T_5!, T_7!, T_{10}?, T_{11}!, T_{13}?, T_{14}!, T_{16}!, T_{18}?, T_{19}!, T_{21}?\}$, while the unobservable ones are $\Sigma_u = \{T_2, T_3, T_6, T_8, T_9, T_{12}, T_{15}, T_{17}, T_{20}\}$. With this configuration, we conduct the verification and get the SOG in Fig. 2(b). Thus, the CPr workflow is not opaque and is not k-step weakly and strongly opaque. Indeed, the two secret states S_{22} and S_{23}, each belonging to an aggregate that doesn't hold other non-secret states. An attacker can then disclose secret information after the traces $T_7T_{13}T_{16}T_{18}$ and $T_7T_{13}T_{19}T_{21}$. The CPr service is therefore unsafe and needs to be improved. Taking into account that the CPub is available for public use, we don't have secrets to be hidden from an external observer. So, we will only describe the CPub actions (Fig. 1(d)) and we won't proceed the opacity verification. The first set of CPub actions concerns the internal operations which include the processing of the data sent by the CPr: the classification (T_2) and the prediction (T_5) which aims to detect the risk of heart attack. As regards the external operations, the CPub receives two messages from the CPr. The first one (T_1) includes the anonymised data and the second (T_4) includes the data collected by the Br and the class to which the patient belongs. In response to the received messages, the CPub sends the classification result to the CPr (T_3) and sends 3 types of alerts according to the prediction results: T_6 for the low alert, T_7 for the medium alert and T_8 for the high alert.

The last service is that of the App (Fig. 1(e)). The set of its internal operations are the notification (T_9) and the application to register (T_1) which allows

a new patient to deposit his information. After registration, the provided information will be sent (T_2) to the CPr. The App shares patient information with the Fog ($T_3!$ & $T_4?$) when this latter communicates for the first time with the Br. It also shares the medical history with the CPr ($T_5!$ & T_6) when it fails to find the patient ID. At the end, the App receives two types of alerts ($T_7?$ for the medium and $T_8?$ for the high) when the risk of a heart attack is detected. The App must be opaque with regards to its set of secret states when dealing with either the CPr or the Fog. To match these needs the observable transitions are $\Sigma_o = \{T_2?, T_3!, T_4?, T_5!, T_6?, T_7!, T_8!\}$, while the unobservable ones are $\Sigma_u = \{T_1, T_9\}$. The set of secret states are $S = \{S_2, S_6, S_7, S_9, S_{10}, S_{11}\}$, with S_2 is related to the request to register a patient, S_6 is that related to sending patient data, S_7 is that triggered due to the sending of personal information of a new patient, S_{11} is related to sending the medical history, and finally S_9 and S_{10} reflect the secrets associated with sending the notification. Conducting the opacity verification, we obtain the SOG depicted in Fig. 2(c). We say that the App SOG is not opaque, and it is not K-step weakly, and strongly opaque.

5 SOG-Based Enforcement of Opacity

In this section, we describe the opacity enforcement problem introducing algorithms to secure the heart attack detection system. Considering a language L and a secret language $L(\varphi) \in L$, when opacity fails of a secret φ for a finite system S, we provide an effective method to synthesize automatically a system S' obtained by minimally modifying the system S so that the secret φ is opaque for S'. To synthesize S', we focus on language modification. If a secret language $L(\varphi)$ is not opaque for a system behavior described by the language $L(S)$, we can modify the behavior by padding it with dummy behaviors. We can then extend the language by computing a minimal super-language of L. In [15], the author has derived an algorithm to compute $min \prod_{super}^{\varphi}$ to assist the designer develop a system that satisfies the opacity property for a secret language.

Theorem 1. *[15] Let a language L defined on an alphabet $\Sigma = \Sigma_o \cup \Sigma_u$ and a static projection π_O defined above on the same alphabet and a secret $\varphi \subseteq L$, then:*

$$min \prod_{super}^{\varphi}(L) = L \cup (\pi_o(\varphi) \backslash (\pi_o(\varphi) \cap \pi_o(L \backslash \varphi)))$$

The proposed approach builds upon the SOG structure to check the system's opacity. If the system is not opaque, the SOG construction allows for detection of all opacity violations provided as a counterexample. These counterexamples will later be used to improve the system security (opacity) by locating the paths leading to the disclosure of private information and performing necessary changes that would render it opaque. Then we compute the minimal super-language that provides us with the restricted language to be added in order to modify the system behavior. For each incident of opacity violation, we match a trace among the calculated super-language and an unobservable event will be added to this trace. In order to opacify the system, we apply the backtracking method. We implement adjustments where needed to the SOG and the LTS and we thus return to the starting model, the Petri net.

5.1 The SOG-Based Algorithm for the Verification of Simple Opacity

The use of SOG-based algorithm in the verification of simple opacity proved efficient [3]. This is due to the symbolic representation of the aggregates, and to the on-the-fly verification. The SOG construction is stopped when the property is proven unsatisfied and a trace (counterexample) that violates the opacity is supplied. To adopt this algorithm for our enforcement approach, we will bring necessary modifications to it.

Algorithm 1. SOG-based Opacification

```
Procedure: SOG-based Opacification ((P, T, F, W), m_o, m_S, Σ_o ∪ Σ_u)
1  Vertices V; Edges E;
2  Aggregate a, a';
3  Stack st, CounterExample;
4  Incidence Matrix C;
5  begin
6      (Q, q_init, Σ, δ) ← BuildReachabilityGraph(P, T, F, W, m_o);
7      S ← m_S;
8      a ← Saturate({q_init});
9      if (a ⊆ S) then
10         CounterExample.Push(ϵ, a, ϵ, a);
11     end
12     V ← a; E ← ∅;
13     trace ← ∅;
14     st.push((a, EnableObs(a)));
15     while (st ≠ ∅) do
16         (a, enb) ← st.Top();
17         if (enb ≠ ∅) then
18             st.Pop();
19         else
20             t ← RemoveLast(st.Top.Second());
21             a' ← Img(a, t);
22             a' ← Saturate(a');
23             if (Treated(a')) then
24                 E ← E ∪ t;
25                 Save(a -t-> a');
26             else
27                 if (a' ⊆ S) then
28                     Trace = Print CounterExample();
29                     CounterExample.Push(trace, a, t, a');
30                 end
31                 V ← V ∪ {a'};
32                 E ← E ∪ t;
33                 Save(a -t-> a');
34                 st.Push(a', EnableObs(a'));
35             end
36         end
37     end
38     if (CounterExample ≠ ∅) then
39         Opacification();
40     end
41 end
```

Algorithm 2. Opacification

```
Procedure: Opacification()
1  begin
2      minSL = ComputationMinSL(L();
3      while (CounterExample ≠ ∅) do
4          (trace, a, t, a') ← CounterExample.Top();
5          if (NotTreated(a')) then
6              foreach u in minSL do
7                  if (u = trace) then
                       /* SOG Opacification */
8                      q_new = new State();
9                      a' ← a' ∪ {q_new};
10                     Save(a -t-> a');
                       /* LTS Opacification */
11                     q ← CounterExample.Top.Fourth();
12                     t_new ← new UnobservableTransition();
13                     Q ← Q ∪ q_new;
14                     Σ_u ← Σ_u ∪ t_new;
15                     δ(q, t_new) = q_new;
                       /* Petri net Opacification */
16                     p_new ← newPlace();
17                     P ← P ∪ p_new;
18                     T ← T ∪ t_new;
19                     p ← getPlace();
20                     F ← F∪(p, t_new);
21                     F ← F ∪ (t_new, p_new);
22                     W ← W ∪ {((p, t_new) ⟼ 1), ((t_new, p_new) ⟼ 1)};
23                     C(p_new, t_new) ← W(t_new, p_new) − W(p_new,tnew );
24                 end
25             end
26         end
27         CounterExample.Pop();
28     end
29 end
```

Taking into account that we are trying to opacify Petri nets, the first modification needed to the algorithm presented in [3] consists in replacing the input by a Petri net-modeled system. The petri net has 2 sets of transitions: observable and unobservable actions, and a set of secret marking subsequently representing the states judged to be secret in the LTS. We add in line 3 a Stack, namely *CounterExample* with all the standard functions (*push*, *pop* and *top*), whose elements are quadruples composed by the counter-examples, a transition t, an actual aggregate a and an aggregate a', successor of a by t. Then, the algorithm 1 starts by constructing (line 6) the reachability graph which represents the LTS. Once other changes have been made (i.e. line 10 & 29), when the opacity is violated, neither the verification nor the construction of the SOG stops. All the paths leading to the disclosure of privacy are stacked into *CounterExample*. Once all nodes are explored and the SOG construction is finished, and if the stack is not empty we proceed to opacification.

5.2 The Opacification Proposed Algorithm

The opacification algorithm has a pretty straightforward mechanism. It begins by computing the minimal super-language. The next step consists in recuperating (line 4) the first elements of the stack (*CounterExample*). Next, the algorithm goes through the *foreach* loop which takes each word of the calculated super-language. If such a word is equivalent with the trace recuperated from the stack, then we proceed to opacify the SOG. We begin by creating (line 8) a new state q_{new} that we will add (line 9) into the aggregate a'. At line 11, we pass to opacify the LTS. We retrieve the last state q included in the aggregate a'. A new unobservable transition t_{new} will be created. Then, the algorithm inserts (line 13) the new state q_{new} to the LTS states, adds (line 14) the new transition t_{new} to the set of unobservable events Σ_u, and defines the transition function between q, t_{new} and q_{new}. Starting from line 16, the algorithm performs the Petri net opacification by creating, at first a new place p_{new} and adding it to the set of places. It also adds the transition t_{new} to the set of transitions. To specify the flow relation between p, t_{new} and p_{new}, the algorithm adds an arc for each relation and assigns to each arc a weight. Afterwards, it modifies the incidence matrix. Finally, the algorithm pops the stack and restarts the operations until the final emptying of the stack presenting the ending test of the while loop.

Being a particular type of Petri nets, oWF-nets require different method of opacification. When fetching the place p (the execution of *getPlace*), we have to exclude the output places. Furthermore, oWF-nets require only one final place p_o. So, following the addition of the unobservable transition t_{new}, we must escape adding the new place. And a flow relation will be added between t_{new} and p_o. Other specific case that may be necessary, when the place returned by *getPlace* is a destination place, we require further changes on the oWF-net. The first step is to retrieve the transition that following its crossing marked the output place. Step two is to delete the flow relation between t and p_o. The following step is to create a new place p_{new} and to add the unobservable transition t_{new}. Then, we

create the flow relations between t, p_{new}, t_{new} and p_o. For the application of the opacification function on our case study, see in this paper [17].

6 Conclusion and Future Work

In this paper, we used opacity, a generalization of many security properties, as a means to track the information flow in an IoT-based medical application. We introduced a model to analyze the behavior of an IoT-based heart attack detection system discussing how an observer may infer personal patient information. Our work aims at detecting security leaks, using SOG-based algorithms for the on-the-fly verification of opacity variants (simple, K-step weak, and K-step strong opacity). We have also proposed a novel, SOG-based approach for opacity enforcement of Petri net-modeled systems. The main contribution of this work is to propose an efficient algorithm for enforcing simple opacity by padding the system with minimal dummy behavior. In our future research, we will explore the same idea of enforcement for other opacity variants such as K-step weak and K-step strong opacity. We also hope to extend this work to take into account different types of enforcement, such as supervisory control for opacity and finding the supremal sub-language, instead of computing the minimal super-language.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
2. Bonomi, F., Milito, R.A., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012. pp. 13–16 (2012). https://doi.org/10.1145/2342509.2342513
3. Bourouis, A., Klai, K., El Touati, Y., Ben Hadj-Alouane, N.: Checking opacity of vulnerable critical systems on-the-fly. Int. J. Inf. Technol. Web Eng. (IJITWE) **10**(1), 1–30 (2015)
4. Bryans, J.W., Koutny, M., Mazaré, L., Ryan, P.Y.A.: Opacity generalised to tansition systems. Int. J. Inf. Secur. **7**, 421–435 (2008)
5. Dubreil, J.: Monitoriting and Supervisory Control for Opacity Properties. Ph.D. thesis, University of Rennes 1, November 2009
6. Falcone, Y., Marchand, H.: Various Notions of Opacity Verified and Enforced at Runtime. Technical report INRIA (2010)
7. Falcone, Y., Marchand, H.: Runtime enforcement of K-setp opacity. In: 52nd IEEE Conference of Decision and Control, pp. 7271–7278, December 2013
8. Li, M., Yu, S., Zheng, Y., Ren, K., Lou, W.: Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE Trans. Parallel Distrib. Syst. **24**(1), 131–143 (2013)
9. Lin, F.: Opacity of discrete event systems and its applications. Automatica **47**(3), 496–503 (2011)
10. Lowe, G.: An attack on the needham-schroeder public-key authentication protocol. Inf. Process. Lett. **56**(3), 131–133 (1995)

11. Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the soa. Ann. Math. Comput. Teleinform. **1**, 35–43 (2005)
12. Mazaré, L.: Using unification for opacity properties. In: Proceedings of WITS (Workshop on Information Technology and Systems), vol. 4, pp. 165–176 (2004)
13. World Health Organization: May 2014. http://www.worldlifeexpectancy.com/tunisia-coronary-heart-disease, consulté le 14/02/2017
14. Saboori, A., Hadjicostis, C.N.: Opacity-enforcing supervisory strategies via state estimator constructions. IEEE Trans. Automat. Contr. **57**(5), 1155–1165 (2012). https://doi.org/10.1109/TAC.2011.2170453
15. Yeddes, M.: Enforcing opacity with orwellian observation. In: 13th International Workshop on Discrete Event Systems, WODES 2016, Xi'an, China, 30 May–1 June, 2016, pp. 306–312 (2016). https://doi.org/10.1109/WODES.2016.7497864
16. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. J. Internet Serv. Appl. **1**(1), 7–18 (2010)
17. Zrelli, R., Yeddes, M., Ben Hadj-Alouane, N.: Checking and enforcing security through opacity in healthcare applications (2017)