Data Minimization in distributed networks for more privacy

September 23, 2016

Abstract

The presence of connected devices in our environment is increasing. These devices form a network often called Internet of Things (or IoT for short), where everything from lightbulbs to thermostats can be controlled by an app or by another device. These services make a lot of that data available to the end user but also to malicious parties due to the devices leaking more data than intended or by bad design. This puts the end user at risk, violating its privacy and leaking sensitive data. One simple and obvious way to prevent leakages and misuses of personal data is to collect less of this data, a principle known as data minimization. However, this solution is rarely used in practice because of business models relying on personal data harvest on one hand and because of the difficulty to enforce it once it is defined what is actually needed to provide a service.

Contents

1 Introduction

1.1 Acknowledgements

1.2 Motivation

1.3 Aim

In this thesis I investigated ways to improve privacy in a special kind of IoT devices known as Wireless Sensor Networks (WSN). WSN are networks of autonomous sensors and actuators. The goal to enhance privacy for this kind of devices will be addressed by relying on data minimisation. This means the project sought to improve privacy in distributed networks by limiting the amount of personal data being processed.

1.4 Scope and Limitations

1.5 Thesis Structure

2 Background

2.1 Wireless Sensor Network (WSN)

2.2 Data Minimization

As defined by the EDPS (European Data Protection Supervisor); "The principle of "data minimization" means that a data controller should limit the collection of personal information to what is directly relevant and necessary to accomplish a specified purpose. They should also retain the data only for as long as is necessary to fulfill that purpose. In other words, data controllers should collect only the personal data they really need, and should keep it only for as long as they need it."

2.3 Related Work

2.3.1 Smart City

3 Theory

3.1 Formal Development

3.2 Model Checking

4 Specification

4.1 Definitions

4.1.1 Actors

A wireless sensor network is built up by several different entities that communicates data between each other. Generally a network consists of multiples of virtually the same entity, e.g. multiple collection nodes, where each of these are running the different instances of the same process. To generalize this concept, the project used a definition called an **actor**. For example, a collection node is an actor since it collects data from an environment and forwards it to a server.

Definition: An actor is a device in the network that communicates data and acts according to a defined behavior.

Formal Definition: Let A be actor (or tuple? or something?) B be a finite set and it's elements are called states of A. Let a D be a device in a network and $P(x, y, ...)$ be a process depending on the variables $x, y, ...$ Let $< D, P >$ denote that process P is running on device D. Then if for each variation of $x, y...$ for $< D, P >$ there is a corresponding state in B, we say A is an actor on D with B as it's behavior.

To describe the interaction between several units in a network, one can specify how they behave in regards to their program state and the input received. Now that we have a general way of describing each group of devices, we can define their behavior in a **Behavior Model**.

image of a behavior model example here

describe the features of the behavior model here and what they do.

4.1.2 Over-collection

Consider a collection process collecting some personal data from a users. The process is only interested in collecting as much data as possible and storing it at a central server for future processing. We can assume that the process will need a good quantity of data entries to extract relevant information from the data. If we assume that the program doesn't remove any data it's collected, we can safely say that at some point the system will have collected enough data to accomplish the tasks it's designed to do and beyond that point it's collected more than it requires.

Definition: Over-collection is the event (state?) when a process collects more data than it requires to function properly.

Formal Definition

from thibaud:

Let a process P be able to collect data and to evaluate boolean expressions. Let a service S(x, y, ...) be a boolean expression depending on variables x, y, ... We say the process P dedicated to the service S, noted ¡P,S¿, over-collects data if and only if P gets any data concerning one of the variable appearing in S after S has been evaluated to be true.

—

4.1.3 Decisions

A decision is a control message sent through the network forcing an action to be taken, in this report that's mainly when over-collection is occuring. This can for example mean that the server is telling one (or several) collection nodes that they should shut down or wait until sending again. If the nodes are also computing data they can still be kept doing so but without sending communication throughout the network until notified again. The source of the decision depends on if the network has a centralized processing unit (f.e. a master-slave relation) or if computations are decentralized and each processing unit makes decision on their own and uses communication to forward processed data to a storage server.

Definition: A message sent in the network is considered a decision if it changes the behavior of an actor.

4.2 Defining the models

As suggested when developing formally, I started out with a simple model as possible. So the model expects that each actor acts as intended, without ma-

licious users or failing procedures, and that all messages gets through, e.g. no 'lossy' channels.

The basic features of the system is a set of collection nodes, a server and an environment.

imagehere

The environment is a simple procedure, it starts off by generating random data and serving it to a requesting node. This is a modeling simplification, since the node should rather extract the data from the environment rather then be waiting for a response. But this was made to simplify the system and made the environment easier to manage as a shared resource for the nodes.

The collection nodes follows the same procedure, where they start by 'requesting' data from the environment. Then the model was split into two different choices depending on the structure of the system. If the system was decentralized, the decision could be taken by a node. Meaning that then the node would first check if the data requested from the environment was below a certain threshold. Then it would send the data along to the server with either a message 'go', meaning that all was fine, or a message 'stop' meaning that over-collection was occurring and it was going to stop collecting and the other nodes should do the same.

If the system was a centralized one, the node would simply pass the data forward to the server. Then it would wait for a response to see if it should continue collecting or if it should stop. This describes a typical scenario for the system, so from this a behavior model for each actor can be defined.

imagehere

The behavior model of the nodes can explain the behavior of the entire centralized system, since the node 'communicates' with all the involved parts and is the one taking (being effected of) all the actions. What distinguishes the decentralized system and the centralized can be noticed in the behavior model for the nodes, in the state marked 'data retrieved'. Here the node would check the data on it's own and notify the server of the action being taken, but the behavior model still works for both cases.

This model expects a perfect behavior from all actors involved, since it doesn't take into account bad behavior (e.g. message-loss or malicious usage) which is not unlikely to occur in a WSN. But this isn't meant to be captured by the model either at this stage, but will be considered when later extending the model.

5 LTL Properties

5.1 Initial Model

The first model had properties for **correctness** and **liveness**. Due to the simplicity of the model, made both of them also rather simple to manage. The correctness property was stated as follows:

When over-collection has occurred (the decision is taken), the system should stop collecting.

Which was translated into:

always (M implies (eventually D))

Where **M** and **D** corresponds to the event that the message is sent and the collection is stopped respectively. Over-collection and the decision were rather interleaved in this system, the decision was taken at the same time the data was checked, therefore the brackets. The liveness property was stated as:

The program shall collect until over-collection has occurred.

Which became:

always (not D until M)

Where **D** and **M** are the same events as described previously.