# Building Private-by-Design IoT Systems

Igor Zavalyshyn
INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
UCLouvain, ICTEAM, Louvain-la-Neuve, Belgium
igor.zavalyshyn@tecnico.ulisboa.pt

## Abstract

Internet of Things (IoT) devices have revolutionized the way we interact with our physical environment. With a single tap on a smartphone screen or a voice command one can control home lighting, thermostats and cameras, monitor physical activity, and keep track of personal belongings. However, while these devices become more and more embedded in our daily lives, there are growing concerns over the privacy and security of highly sensitive data they collect. Numerous cases of data abuse, unauthorized sharing and leakage have been reported. Unfortunately, existing IoT systems have not only failed to prevent such cases, but often contributed to those. To address this issue, we propose a clean-slate approach to building secure and private-by-design IoT systems, in which users retain full control and ownership of their IoT data. The approach builds upon key design concepts: (1) a dataflow programming model for building IoT apps and services, and (2) a mechanism to track sensitive sensor data flows inside these apps and automatically verify their compliance with user-defined privacy and security preferences.

***CCS Concepts:*** • **Security and privacy → Information flow control**; **Access control**.

***Keywords:*** IoT, privacy, data flows tracking and verification

**ACM Reference Format:**
Igor Zavalyshyn. 2020. Building Private-by-Design IoT Systems. In *21st International Middleware Conference Doctoral Symposium (Middleware '20 Doctoral Symposium), December 7–11, 2020, Delft, Netherlands.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3429351.3431750

## 1 Introduction

With the rising popularity of IoT systems, such as Amazon Alexa, Google Home or Samsung SmartThings, and a growing amount and variety of sensitive data they collect, there is still a deficit of mechanisms to ensure user privacy and security. In fact, existing IoT systems often resemble a "*black hole*" that collects sensitive sensor data from the connected devices leaving the end users clueless about further data whereabouts. Unfortunately, this situation has already caused numerous incidents of massive data leackage [7], security breaches [8], insider-related eavesdropping or peeping [1, 6], as well as created opportunities for unauthorized surveillance and forensic investigations [3, 4].

To fill this gap, we propose a clean-slate approach to building secure and private-by-design IoT systems. It aims to empower end users with full control over sensor-collected data, and enable them to specify and enforce privacy and security preferences regarding sensor data collection, processing and sharing. Our approach builds upon two key design concepts. First, we introduce a novel *dataflow programming model* for IoT developers and service providers to create their privacy-aware data processing applications, *apps*. It allows to keep track not only (1) of how the sensor data flows within the app, but also (2) of the semantics of the data as it gets processed inside the app. Our second key design concept enables end users to automatically verify if a given app can violate any of their privacy expectations. To this end, we generate a formal model of the app's elements graph using Prolog predicates, and issue a set of queries to determine the existence of data flows that are in conflict with user-defined privacy policy. A user can, e.g., determine if an app that requires access to a camera feed for motion detection can potentially send raw camera images to a security company instead of the *'motion detected'* events as expected.

We applied these concepts in the design of three systems that span across *local* (home), *cloud* and *mobile* domains. For the local domain, we present HomePad [10], a privacy-aware smart home hub that provides a runtime environment for IoT apps that follow the dataflow programming model to process sensitive sensor data in accordance with user-defined rules. It offers a rich API with an extensive set of built-in elements for IoT app development, and a verification engine for the end users to assess the apps' privacy properties. For the cloud domain, we introduce PatrIoT, a private-by-design IoT platform that extends a dataflow programming model to the cloud. It leverages Intel SGX to prevent unauthorized access to the sensor data when processed at an untrusted cloud environment, and offers the end users an intuitive security abstraction named *flowwall* which allows them to

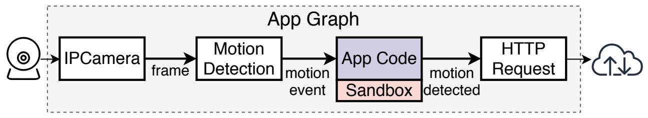

**Figure 1.** Element graph of a MotionAlert app.

specify flexible yet effective policies for controlling sensitive sensor data flows within the apps they install. Finally, for the mobile domain, we propose Flowverine [2], a system for building privacy-aware mobile apps handling sensitive IoT data on Android phones. Flowverine adapts the dataflow programming model to a more complex Android programming and runtime environment, and uses Aspect-Oriented Programming (AOP) for dynamic taint analysis.

Complementary to these three systems, we introduce additional techniques that aim to enhance the security and privacy properties of IoT systems using N-version programming and software hardening. Next, we provide more details on the proposed dataflow programming model, developed systems, and complementary security enhancing techniques.

## 2 Dataflow Programming Model

Our idea of a private-by-design IoT system is one where the end users retain full control over their IoT device data, and can decide themselves on how this data can be shared and processed by various IoT apps and services. Unfortunately, a permission-based access control model used in existing IoT systems can only control the range of devices an app can interact with, and fails to capture apps' data processing and sharing capabilities. As a result, the apps are often overprivileged and may leak sensitive sensor data.

**API based on element graphs:** The dataflow programming model aims to address the problem of permission-based systems. It provides easy-to-use programming abstractions for IoT developers to build privacy-aware apps with all internal data flows made explicit and easy to analyze. The app is implemented as a directed *elements graph*, in which elements represent functional units offered by the system itself (as part of an API) or implemented by the app developer, and the edges describe the only paths through which data can flow within an app. With each element having a well-defined specification, both in terms of interface and expected behavior, such element-based app structure allows for sound and efficient data flow tracking. To illustrate our programming model, consider an element-graph of a MotionAlert app at Figure 1. The app collects camera frames from an `IPCamera` element, forwards those to a `MotionDetection` element which in turn sends motion events to the `AppCode` element whenever motion is detected. The `AppCode` element is provided by the app developer and runs inside a sandbox to prevent direct access to the camera. It sends a motion alert to a security company via `HTTPRequest`. This app package will then consist of a JSON file that describes the app's element graph, and the *JavaScript code* of `AppCode` element.

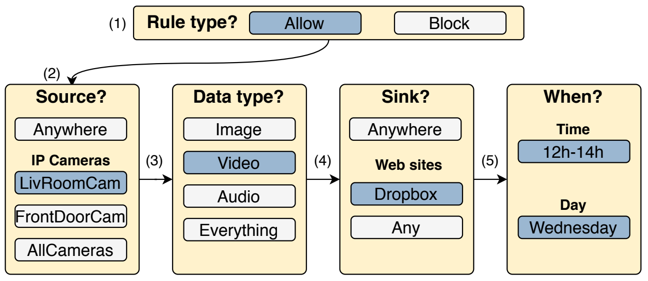

**Figure 2.** Schematic representation of the UI workflow to specify a privacy rule using dataflow model.

**Policy specification and rule syntax:** Since the app described above requires access to both camera and the network, the user will want to make sure that the app cannot leak raw camera images to the security company. To prevent that, he can specify a 'block' privacy policy rule with a particular camera (e.g. LivRoomCam) selected as a source, an `Image` data type, and the `Internet` sink (to denote any web host). More concretely, a privacy policy consists of a sequence of allow / block *rules* which are evaluated sequentially and applied atomically by the underlying application runtime. In general, the format of a rule is as follows:

**`allow`** | **`block`** ⟨data type list⟩ **`from`** ⟨source endpoint list⟩ **`to`** ⟨sink endpoint list⟩ [**`at`** ⟨time period list⟩]

The keywords "`allow`" or "`block`" indicate the *rule type*, i.e., whether the rule allows or disallows the data flows matched by the rule, respectively. The data type list indicates one or multiple types of data to be matched. They can be simple types, e.g., Video, or the wildcard Everything to indicate all possible simple types. The keywords "`from`" and "`to`" are followed by a list of source and sink endpoints, respectively, that may specify individual devices, e.g., LivRoomCam, types of devices, e.g., IPCamera to refer to all IP cameras, or include wildcards, such as Anywhere to refer to any valid endpoint. Optionally, it is possible to specify temporal restrictions by using the keyword "`at`" followed by a time period, e.g., "12:00-14:00", and one or multiple weekdays.

**User interface for policy specification:** To simplify the process of policy rules specification, our dataflow model exposes a simple UI that guides the user along a five step process (see Figure 2) and allows to reason in terms of privacy-sensitive/insensitive data flows the user wants to be blocked or allowed. To create a new rule, he starts by selecting the rule type, i.e., "allow" or "block" (1). Then, he picks the source endpoints of such flows (2), tells what data types from that source he wants to allow or block (3), indicates the destination endpoints (i.e., sinks) (4), and optionally provides a temporal restriction for the rule (5). More sophisticated policies, e.g. based on device state or certain data values, can be supported in the future, but must be carefully conceived as they may increase the complexity of the UI.

**Data flow tracking and verification:** To ensure that a given app abides by the user-specified policy's restrictions, the app's flow graph is then matched against the policy's rules, and the results are reported back to the user. An automatic verification of an app's privacy properties allows the end users to (1) determine what type of information is released by a given app (e.g. to the Internet), and (2) assess at install time whether such a release is acceptable or not. The verification is performed by first creating a model of the app flow graph (named *flow graph model*) using Prolog predicates describing the app elements behavior, connections, and the data types they operate with. The system then queries the app model in order to track the propagation of sensitive data inside the app, and determine the existence of illegitimate data flows, i.e., those that violate the user-defined *privacy policy* rules. For the MotionAlert app, the results would show that only `motion_detected` events are sent to the Internet. The app can then be safely installed.

## 3 HomePad: a Privacy-aware Smart Hub

HomePad [10] extends the architecture of current smart home platforms with the ability to execute third-party IoT apps at the edge. This is achieved by relying on a trusted hub device that can both manage the local devices and provide a platform for executing IoT apps without necessarily depending on the centralized cloud services. As a result, whenever the functionality of an app does not strictly require the shipment of sensor data to the cloud, the data can instead be collected and processed locally, therefore reducing the risks of data exposure and misuse at the remote cloud backend.

HomePad apps follow a dataflow programming model and can be built using a rich set of API elements provided as part of the platform. Using these elements HomePad apps can perform numerous operations, e.g., interact with various sensors and actuators, make network calls, and perform various computations on sensor data (e.g., speech or face recognition, voice synthesis, or data anonymization). The end users may specify app-specific privacy policy rules and verify those at app install time.

When a home app is installed on the hub, HomePad instantiates element objects on the kernel runtime and sets up connections between element instances so as to reflect the flow graph specified in the app package. Each element object can interact with a local system driver which serves the specific requests of that particular element.

We implemented HomePad in Java and used it to test 20 IoT apps in an emulated IoT scenario. We then evaluated HomePad's performance, app programming effort, verification effectiveness, and policy expressiveness. HomePad has low performance overhead (< 6%) compared to standalone app execution, has low entry-level for developers, can successfully detect privacy violations, and its privacy policies can express common users' concerns.

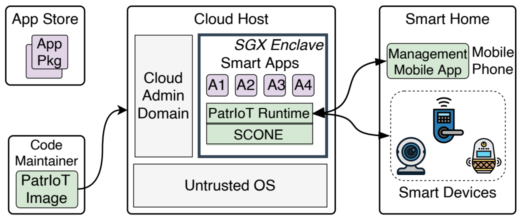

**Figure 3.** System model of our private-by-design IoT platform.

## 4 PatrIoT: a Private-by-Design Platform

HomePad's requirement for local data processing is essential for user privacy, but may be too restrictive for apps that require significant computing resources (e.g., machine learning). To this end, we present PatrIoT which extends dataflow programming model and its privacy guarantees to the cloud environment. Complementary to HomePad's data protection techniques, PatrIoT introduces new security measures that prevent arbitrary access to sensor data by cloud providers or any external attackers.

PatrIoT's architecture is presented in Figure 3. To ensure sensor data confidentiality and integrity protection, PatrIoT runs inside a memory-isolated environment provided by the Intel SGX secure enclaves, and exposes a hardened backend service for managing IoT devices and hosting apps. With a *management app*, the users may securely interact with PatrIoT service and install the apps from an *app store*. The apps run inside sandboxes, and can access sensor data based on permissions and a global user-defined security policy. When PatrIoT is bootstrapped at a cloud host, it runs a remote attestation procedure to obtain proof of integrity and authenticity. The default configuration is further protected by sealed storage encryption and is verified and decrypted only after successful attestation. The user may then start using PatrIoT, connect devices and install apps.

We envision a model where PatrIoT software is maintained by a trustworthy *code maintainer*, which can be a single reputable entity or a consortium, and released open source to help detect potential code vulnerabilities. It can be shipped in the form of a container or VM image ready to be deployed on general-purpose cloud with SGX support (e.g., Microsoft Azure's CC [5]), or be offered as a service by cloud providers to all security-conscious IoT users on a pay-per-use model.

For policy specification PatrIoT introduces the notion of *flowwall*, in analogy to a "firewall", for controlling sensitive sensor data flows generated by third party apps. In contrast to HomePad's verification mechanism, a flowwall implements an information flow control (IFC) model that allows users to: (i) reason about global data flows generated by their connected devices rather than concentrating on individual apps, and (ii) block or allow privacy-sensitive flows without overwhelming them with too much information or level of detail. Flowwall policies are specified once and for all the individual devices or device types, and can be dynamically changed by the user at any time.

We built a prototype of PatrIoT by leveraging SCONE library OS, which allows us to deploy the PatrIoT backend as a Docker container and run it inside an SGX enclave. PatrIoT provides a JavaScript API and runs on top of Node.js. We evaluated PatrIoT by emulating a setup with 10 different IoT devices and 20 IoT apps. We were able to express a range of privacy policies and validate their enforcement in PatrIoT. 45 participants of a usability study found PatrIoT to be easy to use, and its policy rules to be useful for privacy protection. Performance wise, PatrIoT can sustain the typical smart home traffic load, despite a significant SGX overhead (2K inside vs. 9Kreq/s outside SGX), partially caused by the fact that SCONE is not optimized for network-heavy services (other systems, e.g., Graphene-SGX or Occlum might be better alternatives). App execution times are less affected by SGX constraints and are tightly dependent on each app's workload, ranging between 32 and 690 ms.

## 5 Flowverine: Secure IoT Apps on Android

As in smart home systems, leakage of personal data in a mobile context can cause serious privacy breaches. Floweverine [2] addresses the needs of IoT app developers for mobile platforms, e.g., Android, and their respective users. For instance, a fitness-tracking app that reads the user's heart rate from a Fitbit fitness tracker must guarantee that this information can never be shared with unauthorized parties (similarly, e.g., to a contact tracing COVID-19 app). However, ensuring the absence of bugs and security vulnerabilities in the app code is in itself a difficult task, e.g., due to the complexity of the Android API. Furthermore, third-party libraries (e.g., ad libs) included in the app, may have their own vulnerabilities, or, worse, contain malicious code leaking user data. Thus, it is important to have mechanisms that allow both app developers and users to control sensitive data flows within their apps, and consequently block those flows that can lead to security or privacy violations. Unfortunately, despite the security advancements in Android OS, information flow control (IFC) mechanisms are not yet available.

To fill this gap, Flowverine allows app developers to create secure-by-design privacy-aware Android apps. Similarly to HomePad and PatrIoT, it builds upon the dataflow programming model. The apps are structured as a graph of Flowverine-specific elements that mediate access to the Android native API. To prevent the app components from accessing Android API directly, app elements written by the developer run inside sandboxes, and we use Aspect-Oriented Programming (AOP) to intercept native Android API calls and perform dynamic taint analysis. AOP precludes the need to modify the Android OS, thus favoring compatibility.

By ensuring that an app can only generate information flows explicitly declared in the app's element graph, Flowverine helps to prevent security breaches that may result from programming errors or by the inclusion of malicious libraries. Flowverine provides complimentary tooling support for validating the information flows of a given app against an information flow control (IFC) policy using Prolog predicates. Although for different contexts, IFC policies are useful to both app developers and users. An app developer can specify an IFC policy to validate the app compliance with the terms of the service's privacy policy and the data protection rules imposed by law (e.g. GDPR). The user can then perform the same validation or / and apply additional policy restrictions.

Our performance evaluation with a set of Android apps shows that Flowverine has no noticeable impact to the user experience. It was able to (1) prevent hidden sensitive data flows, (2) allow for the strict privilege separation of multiple independent flows within any given app, and (3) support the main Android API programming abstractions.

## 6 Further Security Enhancements

Dataflow programming model relies on a trusted set of elements. Nevertheless, complex elements may potentially contain bugs. To address this issue, we perform an in-depth study to assess whether N-version programming (NVP) can be used to bootstrap trust in these elements [9]. We implemented various privacy-sensitive elements, e.g., face and speech recognition, data encryption and anonymization, and tested them extensively in different N-version settings, and with different merging mechanisms to resolve redundant outputs of multiple element versions (refer to [9] for details). Our study reveals that NVP has a great potential for practical application and is viable to securing IoT software.

Software hardening is another approach used for securing IoT components by adding a safety logic to detect faults and minimize their impact. While there is a variety of hardening techniques, it is hard to predict their impact on software security and performance. We conducted a thorough analysis of common hardening techniques and evaluated their effectiveness in preventing sensitive data leaks, general fault-tolerance, and performance impact [11]. We offer a guideline for IoT developers seeking to make their software secure, and a tool for automatic software fault-tolerance evaluation.

## 7 Conclusions

We presented key design concepts for building private-by-design IoT systems and showed how these concepts can be applied in real-world scenarios. The users of such systems obtain fine-grained control over their IoT devices' sensor data, and can express and enforce their privacy and security preferences regarding data collection and sharing.

**Acknowledgments:** I thank the reviewers for their comments. I also thank my supervisors Nuno Santos, Ramin Sadre and Axel Legay for their continuous support and guidance. This work was partially supported by national funds provided by Fundação para a Ciência e a Tecnologia (FCT), via the UIDB/50021/2020 project, and a CISCO research grant.

## References

[1] Adam Clark Estes. 2018. Yes, Your Amazon Echo Is an Ad Machine. https://gizmodo.com/yes-your-amazon-echo-is-an-ad-machine-1821712916.

[2] Eduardo Gomes, Igor Zavalyshyn, Nuno Santos, João Silva, and Axel Legay. 2020. Flowverine: Leveraging Dataflow Programming for Building Privacy-Sensitive Android Applications. In *Proceedings of 19th IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom) (to appear)*.

[3] Christine Hauser. 2018. Police Use Fitbit Data to Charge 90-Year-Old Man in Stepdaughter's Killing. https://www.nytimes.com/2018/10/03/us/fitbit-murder-arrest.html.

[4] Jay McGregor. 2019. Here's How Amazon's Ring Doorbell Police Partnership Affects You. https://www.forbes.com/sites/jaymcgregor/2019/08/06/heres-how-amazons-ring-doorbell-police-partnership-affects-you.

[5] Microsoft. 2020. Microsoft Azure Confidential Computing. https://azure.microsoft.com/en-us/solutions/confidential-compute/.

[6] Charlie Osborne. 2019. Amazon employees listen in to your conversations with Alexa. https://www.zdnet.com/article/amazon-employees-are-listening-in-to-your-conversations-with-alexa/.

[7] Tara Seals. 2018. Amazon Sends 1,700 Alexa Voice Recordings to a Random Person. https://threatpost.com/amazon-1700-alexa-voice-recordings/140201/.

[8] Amanda Yeo. 2019. Data leak by IoT device maker Wyze exposes personal information of 2.4 million people. https://mashable.com/article/wyze-smart-home-data-leak-breach/.

[9] Igor Zavalyshyn, Nuno O Duarte, and Nuno Santos. 2018. An Extended Case Study about Securing Smart Home Hubs through N-version Programming.. In *Proceedings of ICETE (2)*. 289–300.

[10] Igor Zavalyshyn, Nuno O Duarte, and Nuno Santos. 2018. HomePad: A privacy-aware smart hub for home environments. In *Proceedings of The Third IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 58–73.

[11] Igor Zavalyshyn, Thomas Given-Wilson, Axel Legay, and Ramin Sadre. 2020. Brief Announcement: Effectiveness of Code Hardening for Fault-Tolerant IoT Software. In *Proceedings of 22nd International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS) (to appear)*.