Z B. Celik, Earlence Fernandes, Eric Pauley, Gang Tan, Patrick McDaniel

Penn State University, University of Washington

A Program Analysis of the Security and Privacy of Home-based IoT Applications and Systems

Denise Paredes, Matthew Zaldaña, and Nicolas Zona

College of Engineering, California State University  Long Beach  
CECS 327, Sec 02, Spring 2022  
Instructor: Malik Luti  
April 30, 2022

Abstract

Thanks to COVID-19, there exists a myriad, exponentially larger, amount of IoT devices connected among themselves in homes around the world. However, given the advances in applications and the closing bridge of knowledge that stems from the Internet, these devices not only connect those between a home and a service, but a network of interactions also now exists between any one IoT physical device and a digital service that wasn’t meant to interact with the device before. Given the scope of these connections, the risk that security and privacy thresholds may succumb have grown as well. Yet, the IoT field is still emerging and while rules are put in place to allow certain controls over devices and the amount of data that they handle, there isn’t one single technique that governs them all nor a foundational method that yields solutions to the discovery of security and privacy violations. Z. Berkay Celik, Earlence Fernandes, Eric Pauley, et al. propose several program analysis techniques that could solve these issues endorsed by the study of the underlying architecture of IoT applications. They present significant challenges that pose an important threat to devices all over the world as well as motivation to solve these issues. One issue I found most pressing in IoT devices was that of the challenge physical channels present when running program analysis to catch unintended functional tasks a device can take. Human nature is to interact with the material objects around us, however, when creating a network of interconnected devices, the intended behavior for these may not result functionally viable to allow its continuance. With the aid of the paper, I will be presenting the researchers proposed solution as well as the other topic considered.

*Keywords*:  Security and privacy - software and application security; Software and its engineering - Automated static analysis; Dynamic analysis; IoT security and privacy, IoT programming platforms, program analysis

**The Sensor-computation-actuator idiom**

Celik et al provided an overview of IoT system architectures to clearly explain how devices are connected in the physical environment. This structure is often developed using a bottom to top basis, “with (1) devices, (2) connectivity protocols, and (3) IoT programming platforms” (Celik et al, 3). This reminds me much of the OSI model where the physical devices are at the bottom layer and the application layer is at the top. It must be understood that a sensor is anything that “can gather information about its environment” (Celik et al, 4). The fundamental working knowledge of sensors is crucial to understanding how the physical layer works with the programmed digital layer. Known as events, applications process interactions between the physical environment and subscribe to handlers that handle an event or action. These then actuate or control the next device in the routed connection, whether that is of physical or digital composition, as can be seen in the figure. Knowing this, Celik et al inspected various domains of programming platforms and these web-based tools “enable developers to write applications used to create custom automations” (Celik et al 4). While applications themselves have their own type of security in place to catch malicious actions, the overall infrastructure, Celik et al argues still has major vulnerabilities that have not been addressed. This happens because of exploiting sensor event handlers and their physical counterpart reactions. For example, one example used a lot is that of a smart light which turns on after the sensor senses the event that someone is in the room. This could allow a burglar to know when a person is in their home and when they aren’t or when they are not moving. Of course, while the application is working as intended, security and privacy are breached in a side channel. Situations like these among others are explored and given small solutions to motivate their creator’s interest in fixing the issues so that IoT world may function smoothly, at least within the security and privacy domain.Diagram

Description automatically generated

**IoT Programming Platforms**

Celik et al looked at five platforms that take up the largest market share: “Samsung’s SmartThings, OpenHAB, Apple’s HomeKit, Android Things, and Amazon AWS IoT” (Celik et al, 5). They inspected each and found that the “IoT platforms use similar programming structures and the differences lie only in the communication protocols between IoT devices and edge systems” (Celik et al, 5). Forthwith, a brief discussion of each of these systems and their operations backed by Celik et al.Developed in the Groovy language, these IoT devices consist of a cloud back-end, a hub which controls communication, and the app which governs the actions between the hub and is run in the cloud. The devices are strictly bound to permissions, subject to actions and events. An action controls the device, and the event is looked for and triggered after the device state changes for a particular event.Built in Java, OpenHAB is all about automation. At its core, rules govern triggers that react to changes. These could be any kind of triggers; however, it is important to note that the “rules are written in a Domain Specific Language (DSL) based on the Xbase language” (Celik et al, 6). Apple’s Swift language is key to this development kit. The HomeManager classes and its child classes form objects that virtualize a location, which represents a home or room. Each accessory that can access this kit is given basic data types such as a Boolean known as a characteristic. Accessing these fields is what is needed to write the methods that control the Apple devices into reacting a certain way. The AWS cloud and connected devices is all that is needed to save an abstract device state in the AWS cloud itself and manage its behavior. However, when connecting deployed AWS apps, the control of devices is infinite. AWS uses HTTP requests and SDKs to wrap these requests to develop apps in Java and C. Furthermore, development includes SQL-like rules, “which are used for filtering messages sent to AWS IoT Core” (Celik et al, 7). These rules can use data from any of the connected AWS devices in that direct connection. Android based apps rely on the Android stack and the Things Support Library which “creates an activity as the main method in its manifest file when the device boots” (Celik et al, 7). Active app listening is necessary for the rest of the process to continue as device states change over the use of the application which then triggers an event. Nonetheless, these IoT applications rely heavily on the event-handlers that govern their next action. These then present situations where program analysis must be taken into consideration.

**Program Analysis**

Celik et al went over the four goals of program analysis and the two types of program analysis that exist currently. Given its shortness, I will also be brief. The researchers prioritized controlling sensitive data leaks which is app data sent to the developer for further inspection, such as when a bug causes an application to crash; abuse prevention that ensures applications only work for its intended purpose and conforms within the user permissions and developer expressed requirements; permission misuse which restricts the permissions the application has to access other device information and functions; and data provenance - my personal favorite - which aims to find where data originated from by tagging origins of data with a flag for easier debugging and bug finding (Celik et al, 9, 10). At the heart of program analysis lies static and dynamic analysis where the end goal is the same for both: analyze a program to mitigate security risks (in this scope) yet achieve some success in the aforementioned goals. Different structures allow for different results - static analysis analyzes code during runtime, meaning no code is injected into the original database and high-level simulations are run to find every path feasible for a particular function, whereas dynamic analysis is code that is run under observed conditions, usually in a sandbox and is injected into the codebase for a particular function or delegate event handler requiring inspection.

**Issues and challenges**

Due to the event-handling nature that IoT devices possess, the researchers can perform analysis on the applications that they surveyed given that other security problems cannot be mitigated by use of program analysis. There are several issues, each unique to IoT security issues. For one, IoT devices control physical hardware with the use of event-driven handlers in different languages, and each language has its own nature, which makes it difficult to perform program analysis. Second, more than one device may exist in a location and they could work together giving the problem of co-located interaction which results in unexpected behavior. Events could trigger events in these cases. Another issue is that trigger-actions connect events of IoT devices with digital services which makes the task easier for the user but the complexity of analyzing the different aspects of security much harder. Lastly, idiosyncrasies that require special treatment due to what the event can do in light of another action due to the programming language (Celik et al, 11). The analysis itself also possesses certain issues that can make it difficult to yield specific results in order to achieve maximum rendering of beneficial goals.

While there are several, I will be focusing on one in particular and the solution proposed by the researchers. Physical channels are a challenge in program analysis because there are many physically connected devices co-existing in the same room at the same time. Not only this but several times these devices are connected to digital services as well. This presents a security issue as “(1) data leaks through side channels, (2) health-related risk through device functionality misuse, and (3) safety issues through indirect physical interactions” (Celik et al, 12). The solution presented is simple: create a template that defines the secure and the unsecure device states for (1) and (2). In fact, extending the solution brings us to the (3) solution, which would be to add “additional path conditions to guard device actions based on the app’s context” (Celik et al, 13). There are other issues with program analysis itself such as simulation and variable constraints in a sandboxed environment, limitations in the automation of test-case generations, the problem of multi-app usage analysis and its interoperability, and the interaction between IoT devices and trigger-action platform services. Each of these has its own proposed or motivated solution called program analysis sensitivities explained in detail, however, again, I will be focusing on one.

Provenance tracking program analysis goes hand-in-hand with the problem of physical channels blurring the view of physically connected devices and data origination because it works in the following way: it tracks the origin of code to see what its intent was and labels the variables as it goes. This is similar to what code forensics does as it aims to find the origins of a malicious program. This way, the program code is found to be either of a developer or user origin or if it is of external sources and could be potentially malicious. The following figure presents an example code block for predicate analysis and provenance tracking. We see how a threshold evaluates a user temperature defined variable and whether a message is sent to the user. What’s important to note are the constants that are hard-coded by the developer and should be flagged as so. However, the “x” variable in line 6 is evaluated given a user threshold which should be labeled as so (Celik et al, 19). Anything else could be flagged as malicious as its intent is only what is in this code, thus finding a security flaw and mitigating any other action that could derive from it. Graphical user interface, text, application, email

Description automatically generated

**Conclusion**

The researchers tried to present a paper that aims to find how program analysis could help abate the risk of violating security and privacy principles in the IoT field. While their work was mostly a survey of systems already in place, it provides a useful insight as to what future work could bring and the changes that we may see in future years.

References

Last Name, F. M. (Year). Article Title. *Journal Title*, Pages From - To.

Last Name, F. M. (Year). *Book Title.* City Name: Publisher Name.

Footnotes

1[Add footnotes, if any, on their own page following references. For APA formatting requirements, it’s easy to just type your own footnote references and notes. To format a footnote reference, select the number and then, on the Home tab, in the Styles gallery, click Footnote Reference. The body of a footnote, such as this example, uses the Normal text style. (Note: If you delete this sample footnote, don’t forget to delete its in-text reference as well. That’s at the end of the sample Heading 2 paragraph on the first page of body content in this template.)]

Tables

Table 1

[Table Title]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Head | Column Head | Column Head | Column Head | Column Head |
| Row Head | 123 | 123 | 123 | 123 |
| Row Head | 456 | 456 | 456 | 456 |
| Row Head | 789 | 789 | 789 | 789 |
| Row Head | 123 | 123 | 123 | 123 |
| Row Head | 456 | 456 | 456 | 456 |
| Row Head | 789 | 789 | 789 | 789 |

Note: [Place all tables for your paper in a tables section, following references (and, if applicable, footnotes). Start a new page for each table, include a table number and table title for each, as shown on this page. All explanatory text appears in a table note that follows the table, such as this one. Use the Table/Figure style, available on the Home tab, in the Styles gallery, to get the spacing between table and note. Tables in APA format can use single or 1.5 line spacing. Include a heading for every row and column, even if the content seems obvious. A default table style has been setup for this template that fits APA guidelines. To insert a table, on the Insert tab, click Table.]

Figures title:

Figure 1. [Include all figures in their own section, following references (and footnotes and tables, if applicable). Include a numbered caption for each figure. Use the Table/Figure style for easy spacing between figure and caption.]

For more information about all elements of APA formatting, please consult the APA Style Manual, 6th Edition.