Tyche: A Risk-Based Permission Model for Smart Homes

Amir Rahmati
Samsung Research America
Stony Brook University

Earlence Fernandes
University of Washington
University of Michigan

Kevin Eykholt
University of Michigan

Atul Prakash
University of Michigan

***Abstract*—Emerging smart home platforms, which interface with a variety of physical devices and support third-party application development, currently use permission models inspired by smartphone operating systems—the permission to access operations are separated by the device which performs them instead of their functionality. Unfortunately, this leads to two issues: (1) apps that do not require access to all of the granted device operations have overprivileged access to them, (2) apps might pose a higher risk to users than needed because physical device operations are fundamentally risk-asymmetric—"door.unlock" provides access to burglars, and "door.lock" can potentially lead to getting locked out. Overprivileged apps with access to mixed-risk operations only increase the potential for damage. We present *Tyche*, a secure development methodology that leverages the risk-asymmetry in physical device operations to limit the risk that apps pose to smart home users, without increasing the user's decision overhead. Tyche introduces the notion of risk-based permissions for IoT systems. When using risk-based permissions, device operations are grouped into units of similar risk, and users grant apps access to devices at that risk-based granularity. Starting from a set of permissions derived from the popular Samsung SmartThings platform, we conduct a user study involving domain-experts and Mechanical Turk users to compute a relative ranking of risks associated with device operations. We find that user assessment of risk closely matches that of domain experts. Using this insight, we define risk-based groupings of device operations, and apply it to existing SmartThings apps. We show that existing apps can reduce access to high-risk operations by 60% while remaining operable.**

I. Introduction

Smart home platforms play the role of connecting heterogeneous devices and protocols while supporting third-party application development. Several such platforms are emerging, including Samsung SmartThings [2], Apple HomeKit [4], and Google Home [1]. These platforms enable the promised home IoT benefits of better convenience, improved security, and more energy efficiency. However, such platforms pose security threats—Fernandes *et al.* performed an empirical evaluation of Samsung SmartThings, and discovered that its permission model automatically overprivileges apps [9]. For instance, an app that locks doors after 9PM also gains the ability to unlock those doors at any time. Although overprivilege has traditionally plagued several types of computing models, in the context of IoT, it can cause physical and material harm, beyond the classically prevalent digital harm.

In this paper, using the popular and widely-used Samsung SmartThings as an example of a prototypical smart home platform, we explore the design space of permission models for smart homes with the goal of reducing the risk posed to users if their smart home apps are compromised or are malicious. In SmartThings, overprivilege occurs because its permission model groups device operations based on functional similarity—"door.lock" and "door.unlock" both control the state of the lock. Functional grouping of operations helps reduce the cognitive burden on developers, when requesting permissions, and on users, when making security decisions during application installation. A very fine-grained model, where apps request permission for each device operation, only increases the burden on developers and users and has been shown to be impractical even in the smartphone environment [8]. The increased number of decisions a user has to make in a smart home environment makes this approach even less feasible. Therefore, despite the usability advantages of a functionally grouped permission model, attackers can use compromised or malicious apps, which are overprivileged, to cause security and privacy risks.

We consider a middle ground between very fine-grained per-device-operation permissions and functionally grouped permissions. Our insight is based on the fundamental risk asymmetry of device operations—a property unique to physical devices. As an example, "oven.on" is a potential fire hazard, "oven.off" is potentially uncooked food, "mic.on" is a privacy risk, "mic.off" might only disable certain voice-assistant functionality like Amazon Alexa, "door.unlock" is a potential burglary risk, "door.lock" only locks the occupant outside (or inside) and is a potential annoyance. We present Tyche, a secure development methodology that leverages this intuitive risk asymmetry, and groups physical device operations into equivalence classes of risk. In this risk-based model, apps specify access to device operations in terms of user-perceived risk-levels. Our goal is to retain the usability advantages of operation grouping while limiting the risks of overprivilege.

Tyche breaks down device functionalities into 3 risk categories (high, medium, and low). For our door lock example, "door.unlock" will be in a high-risk group, and "door.lock" will be in a medium-risk group for the door's operations. Developers request permissions by specifying the type of the device, and the risk category. So, an application that automatically locks the door after a specific hour will request for the permission "door.medRisk" and this gives it access to "door.lock." If the application requests "door.highRisk," it will gain access to "door.lock" and "door.unlock." That is, there is an ordering relation between the risk categories. Intuitively, if the user is willing to give an application high-risk access to a device, the user is not concerned about low-risk access. The risk-based model is an improvement over

978-1-5386-7662-2/18/$31.00 ©2018 IEEE
DOI 10.1109/SecDev.2018.00012

functional grouping because it still groups operations, but in terms of risk. At the same time, Tyche does not prohibitively increase the number of decisions user has to make, but ties an intuitive risk factor to them. Therefore, it bounds the damage that a compromised application can cause, and accurately communicates that potential to users at installation time.

In designing Tyche, we overcame several challenges: (1) We need a methodology to estimate the user-perceived risk of device operations. To that end, we introduce a survey instrument inspired by the methodology of Felt *et al.* that is designed to compute a relative ranking of user-perceived risk[8]. The survey captures concrete risks that can arise from a variety of smart home devices, and then presents those risks to users. We use this methodology to obtain a ranking of user-perceived risk for Samsung SmartThings device operations. We focused on this platform because of its wide support for smart home devices (more than a 100). Furthermore, independently of the specific framework, physical device operations tend to remain the same. Therefore, our risk results are potentially applicable to other smart home platforms as well. (2) We need a technique to form equivalence classes of risk using the survey data. To that end, we utilize a clustering-based approach, and found it to work reasonably well in practice.

Contributions.

- We propose Tyche, the first risk-based permission model for smart home platforms that leverages our insight that physical device operations are fundamentally risk-asymmetric.
- We compute the user-perceived risk of device operations in the Samsung SmartThings platform using a user study of three domain experts and 400 Mechanical Turk participants. This dataset is available at *https://github.com/Ethos-lab/datasets*.
- We empirically demonstrate that user-perceived risk of device operations closely matches that of the experts, motivating future research on permission prompt UI designs that focus on incorporating risk indicators.
- Through a case study of three existing smart home apps, which we port to our risk-based permission model, we establish that the number of risky operations an attacker can issue reduces by 60%.

We envision that Tyche, including its user research methodology and permission model design, will serve as a set of guiding principles for developers of future smart home platforms.

II. Background

A. SmartThings Framework

Figure 1 shows an overview of the SmartThings framework. It consists of: (1) a physical hub that users place in their home, (2) a proprietary cloud back-end that runs third-party applications, and (3) a companion application that runs on a user's smartphone that is used for local control and configuration. This smartphone app serves as a display device for the hub, as the hub itself has no display. Each hub supports a variety of home automation protocols such as ZWave and ZigBee. The companion application lets users download and install third-party SmartThings apps (or *SmartApps*) into their SmartThings cloud account. The SmartApps, written in the Groovy programming language,[1] interact with physical devices by issuing method calls (*e.g.,* lock a door) and by listening to events from the devices (*e.g.,* motion was detected). SmartApps are published to an application store. Physical devices are represented by *device handlers* or *SmartDevices*—pieces of code that use protocol-level instructions to communicate with devices. A device handler wraps a physical device, and exposes it to the rest of SmartThings. Although the platform provides device handlers for a wide variety of devices like door locks and speakers, developers can also write their own device handlers for unsupported devices.

SmartThings Permission Model. SmartApps must request capabilities (permissions)[2] for devices before they can interact with them. A capability is composed of a set of commands (method calls) and attributes (properties). Commands represent ways in which a device can be controlled or actuated. Attributes represent the state information of a device. For example, capability.lock contains the attribute "lock," which represents the current state of the lock, and the commands: lock(), and unlock(), that an application can use to control the door lock. When a user installs a SmartApp, SmartThings triggers the device enumeration process where it lists all physical devices that support the requested capabilities. For instance, if a user's home has two motion sensors, and a SmartApp is requesting "capability.motionSensor," then SmartThings will list both sensors in a permission granting UI. At that point, the user will decide whether to give the application access to the motion sensors or not.

SmartThings supports the functional grouping model—functionally similar device operations are grouped into a single capability. Third-party apps gain access at the level of such functionally-grouped capabilities. Depending on the app's functionality, this model can potentially lead to *automatic* overprivilege [9]. Fernandes *et al.* empirically evaluate the consequences of such overprivilege in their recent security analysis of SmartThings [9].

B. Threat Model

We assume that SmartApps can be malicious or can be compromised. Fernandes *et al.*, in their recent security analysis of the SmartThings platform, demonstrated attacks of both types [9]. The attacks depend on two factors: (a) users accept permission requests without realizing the risk an app poses; (b) apps can ask for more permissions than they need, or they automatically obtain more permissions than they need due to the design of the platform and the granularity of permissions. Under this assumption of overprivileged apps, our goal is to reduce the risk such apps pose to users. We

[1]Groovy compiles to Java bytecode

[2]Although SmartThings uses the word "capabilities," they are in fact permissions and are not related to capability-based security. We will use "permission" in the rest of the paper.

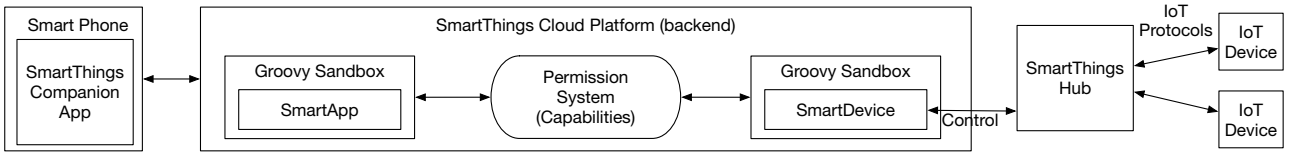

Fig. 1. Samsung SmartThings Architecture.

consider other types of attacks to be outside the scope of this work. For example, a compromise of the platform itself will render permissions themselves useless. Orthogonal techniques are applicable to secure other parts of the smart home platform.

III. Tyche: The Risk-Based Permission Model

Our key insight is that the risk asymmetry between functionally-related operations in smart homes creates an imbalance between the level of access to a device that an app needs, and the level of access provided to it by a traditional functionally-grouped access control system. Permission grouping is vital from a usability standpoint. In theory, we could require that the application request access for each operation of a device. However, this extremely fine-grained system creates a poor user experience, and it ultimately results in users making poor security decisions due to fatigue [8]. We need to group operations together, but in the context of the smart home, this grouping can increase the risks users face.

Tyche avoids this risk by assigning risk levels to each of a device's operations, and then groups operations of the device that have the same risk level into a group. An app can request access to a device's operations by specifying a risk level. The result is that we now have a way of communicating the risk of an app to the user, and we have a way of limiting the risk a compromised or malicious app poses to a user.

There are three design challenges that we need to overcome to design a risk-based permission model: First, how many risk levels should exist? In the simplest case, we could assign a risk level to each device operation. However, this can lead to the same decision fatigue that plagues extremely fine-grained access control systems. We believe that the answer to this question depends on the specific smart home platform, and the specific types of devices it supports. In the context of SmartThings, we experimented with three levels of risk (low, medium, high), and found it to be effective. §IV contains an evaluation showing that with three levels of risk, existing apps can still function, but now with 60% less access to high risk operations. Furthermore, as SmartThings supports more than a 100 types of devices covering a wide range, we envision that the results here are applicable to other platforms that work with similar devices.

Second, how do we create equivalence classes of device operations, grouped by risk level? We envision such breakdown of risk to be done by smart home platform developers. To implement risk-based access control in current SmartThings platform, however, we asked three researchers with expertise in security of Internet of Things platforms to systematically evaluate every function currently available and put them in three risk categories. We averaged experts evaluations and use them as gold standard to group operations based on their risk.

Third, how can we implement the risk levels on the SmartThings platform to concretely demonstrate their value? It is a closed-source system, and developers of SmartDevices (*i.e.,* device handlers) cannot define new capabilities. The straightforward way of implementing a risk-based system is to define new capabilities based on risk levels. However, this is not possible with SmartThings. To overcome this challenge, we adopt the approach used in ContexIoT [11]. We propose automatically injecting security code into the apps at the source code level so that they communicate their risk level to users when they are installed. The remainder of the section details this process. It discusses our techniques by assuming a certain splitting of device operations. §IV contains details on the exact splitting of SmartThings permissions into risk-based permissions. Our goal here is to discuss the methodology of enforcing risk-based permissions on SmartThings apps independently of a specific splitting of device operations. We stress that a risk-based permission model can be enforced in multiple ways. If a smart home platform were to allow customization (*e.g.,* analogous to Android OS modifications in smartphone research), then the platform itself could be modified to support risk-based permissions.

A. Developing a Risk-Based Permission Model

As discussed in §II, each SmartDevice in SmartThings can expose multiple capabilities (or permissions) depending on the kinds of operations the device supports. For example, a door lock will support operations related to battery management, lock control, and lock code programming. For each of these functional categories, there are associated permissions. Tyche retains this model, but introduces a modifier that app developers need to use while requesting access to a device.

Example. Consider a battery management app that requests access to battery-backed devices around the home. Currently, such an app would make a permission request using the following line of Groovy code:

$$input\ "battery\_device", "capability.battery"$$

If the user accepts the permission request, the app will gain access to all operations that are functionally grouped together inside capability.battery. As per our discussions, these operations may differ from a risk perspective. Under Tyche, this app would instead specify a risk level as follows:

$lowRiskRequest$
$input\ "battery\_device",\ "capability.battery"$

The first line is a method call that signifies that the next line of code is a permission request for low risk access to devices that support capability.battery. If a user accepts the eventual permission prompt, the app will gain access to all low risk operations defined by capability.battery. Thus, Tyche keeps code changes to a minimum to help app developers move to a risk-based permission system. We envision that smart home platform developers could adopt a similar technique of using a single modifier for requesting risk-based access.

B. Enforcing a Risk-Based Permissions Model

As the SmartThings platform is closed-source, Tyche utilizes an app rewriting approach for enforcing our risk-based permission model. This is a common approach to enforcing security primitives in SmartThings. Notably, the platform itself performs rewriting to create a sandbox around apps (by allowing and denying certain operations) [9]. Furthermore, recent work on enforcing contextual security policies on SmartThings uses the same approach [11]. Rewriting occurs at the source-code level by operating on the Abstract Syntax Tree (AST) of the program. Groovy supports compiler extensions that perform rewriting of the AST. Our high-level approach is to: (1) Introduce a development-time annotation that the developer uses to specify the risk level of the access being requested; (2) a runtime reference monitor that the compiler injects into the app. This reference monitor hooks all device operation invocations, and verifies that the operation being invoked belongs to the risk level that was requested earlier. Figure 2 shows an architectural overview.

As input, the compiler extension takes app source code where the developer has annotated all permission requests with the risk rating. We envision this to be achievable with minimum developer effort as each permission is only requested once in each application. If there is a permission request without an annotation, the extension will throw a compilation error, guiding the developer to correctly annotate the permission request. The extension also takes in a risk table, that is simply a mapping of SmartThings capability names, operations (commands or attributes), and the risk rating. As we discuss in the next section, we use a user study to compute the risk rating. The extension then performs two steps: (1) it instruments all method invocations on objects that represent physical devices to first consult a reference monitor; (2) it injects a reference monitor into the app that is executed on every method invocation. Its job is to consult the risk table and determine if a particular invocation matches the previously requested risk rating. In our example, if the app attempted a high risk operation, the reference monitor would deny that because the developer had only requested low risk access.

To communicate the risk level of an app, our extension also injects startup code into the app. This results in a notification to the user of the risk levels that the app requested. The notification is shown on the corresponding smartphone app of SmartThings.

TABLE I
RISK-BASED PERMISSION EXAMPLES. COMMANDS ARE SHOWN AS $cmd()$ WHILE ATTRIBUTES ARE PRESENTED AS $attr$.

Permissions	Low-Risk	Medium-Risk	High-Risk
lock	–	lock()	unlock(), lock
alarm	–	strobe(), siren(), alarm	–
switch	switch	–	on(), off()

IV. USABILITY ANALYSIS

We evaluate Tyche from two aspects: First, to understand the usability of Tyche, we evaluate user perception of risk in comparison with risk assigned by the domain experts by conducting a 2 stage user study using 210 participants. Second, we evaluate three applications under the Tyche model. Our results show that Tyche limits access to 60% most high-risk operations, while neither decreasing the functionality, nor increasing user decision overhead.

A. User Study Setup

To understand user perception of risk associated with physical device operations, we surveyed 400 Mechanical Turkers.[3] The survey started out by collecting basic demographic information. Per our IRB, we were not allowed to collect any user-identifiable data. The demographic information we collected are as follows: (1) Participant's age, (2) Number of people in their household, (3) The home platform they use, and (4) In cases where the participant's use SmartThings platform, the number of applications they have installed.

Next, the survey presented a description of the main task:

> *Imagine you have several apps installed in your home automation system. These programs require certain permissions and can perform actions on your behalf. Based on each action, indicate your level of concern based on the following Likert scale:*
>
> 1) ***Not Concerned*** *- Select this for actions which pose little to no risk to you or the people living with you.*
> 2) ***Mildly Concerned*** *- Select this for actions which pose a low risk to you or the people living with you, but are useful when used as intended.*
> 3) ***Concerned*** *- Select this for actions which pose a medium risk to you or the people living with you, but are useful when used as intended.*
> 4) ***Very Concerned*** *- Select this for actions which pose a high risk to you or the people living with you, but are useful when used as intended.*

[3]We received exemption from our institution's IRB under title 45 CFR 46.101.(b). Survey data was collected anonymously and did not include any user-identifiable or sensitive information about the participants.

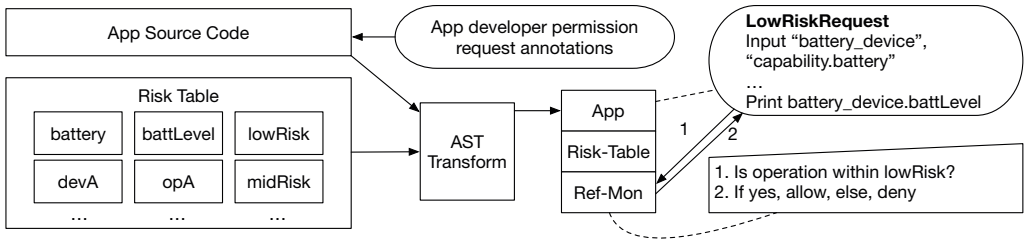

Fig. 2. Tyche rewriting architecture to enforce risk-based permissions.

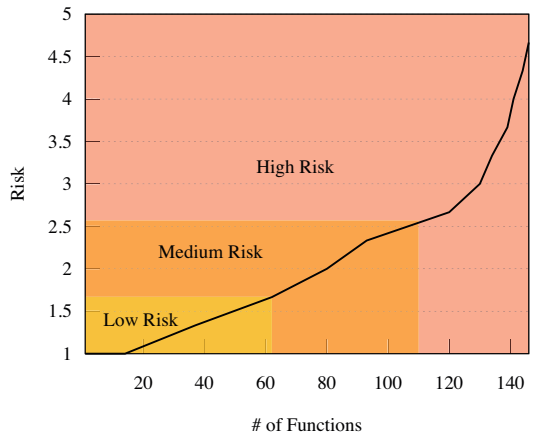

Fig. 3. Cumulative distribution of risk for the 146 SmartThings functions assigned by domain experts. The distribution is divided to three ranges using K-mean clustering and is used as a basis for our risk-based permission model.

5) ***Would Not Allow*** *- Select this for actions which pose a high risk to you or the people living with you and are better left disabled.*

We provided participants with a subset of device operations and asked them to rate their concern based on the above 5-point scale. We varied the information associated with the list of device operations based on the type of study condition. We had the following conditions:

- **Uninformed User Ranking:** We recruited 200 users through Amazon Mechanical Turk to assess the risk of operations based on an *operational description* only (Table II). To reduce decision fatigue, each user was provided with half of the device operations. In total, we obtained 100 ratings per function from these users.
- **Informed User Ranking:** Similar to uninformed user study, we recruited 200 users through Mechanical Turk. In addition to an *operational description*, we provided each user with a *misuse scenario* (Table II) to illustrate the potential risk associated with the device operation. Similar to the previous condition, we divided operations into two groups to reduce decision fatigue and collected 100 ratings per device operation.

As a baseline for our study, We used the domain expert evaluation of all operations from Section III. We consider this data as the gold standard for our system and compare other conditions to it.

Dealing with Random Clickers: Although prior studies have shown that results collected through crowdsourcing approaches are comparable to lab studies [12], our study is still susceptible to random clickers who answer the questions randomly to get the monetary benefit with minimum effort. To minimize this risk, we used our demographic questions as indicator questions to remove random responders (*e.g.,* family size = -2) and the Pearson's χ^2 test to weed out random clickers as suggested by prior work [12]. These filters left us with total of 116 and 94 data points for our uninformed and informed condition, respectively.

B. User Study Results

The age distribution of our participants is presented in table III. The population of our study consists of mostly young adults. We discuss how the age might affect user perception of risk in Section V-B. Table IV also presents the number of apps SmartThings users install in their platform. Our results show that most smart home users stick to very few specific applications. To assess the risk of each operation, each participant provides a Likert scale response (ranging from 1-5) as the risk of each operation. We average the score given by each group of our participants (uninformed, informed, and expert) and use K-means clustering (K=3) to divide these operations into three groups based on their risk. Table V provides an overview of the K-means clustering result and the number of operations falling into the low-, medium-, and high-risk groups. When users consider the risk of an operation, they assign conservative risk values compared to domain experts. Informing the user about the actual misuse scenario lowers the perceived risk of some of the operations, but does not make the user complacent when compared to the expert.

To examine the correlation between the risk assessment of experts (our gold standard) and users, we calculate the Pearson product-moment correlation coefficient (Pearson's r) between the experts and both uninformed and informed users results. Our results show moderate correlation (0.6) between experts and uninformed users and strong correlation (0.75) between the experts and informed users. These results motivate that a risk-based permission system can push users to make better risk assessments of access control requests from apps.

C. Case-Studies: Converting Apps to the Risk-Based Model

We first compute the set of device operations accessible to three existing SmartApps based on the current SmartThings

TABLE II
EXAMPLES OF THE OPERATIONS ASSOCIATED WITH A PERMISSION, THEIR DESCRIPTION, AND THEIR POTENTIAL MISUSE SCENARIO.

Permission	Operation	Operational Description	Misuse Scenario
Switch	switch	Track the status of the switch	Publicly share when switch is turned on/off
Switch	on()	Turn on connected devices	Turn on a device, such a space heater, for a long time, possibly causing a fire
Lock	lock	Track the status of the lock	Publicly share when your door is unlocked
Lock	unlock()	Unlock door lock	Automatically unlock doors without your knowledge
Lock	lock()	Lock door lock	Automatically lock doors without your knowledge

TABLE III
AGE DISTRIBUTION OF USER STUDY PARTICIPANTS.

Age	$18-25$	$25-35$	$35+$
% of Participants	24%	40%	26%

TABLE IV
NUMBER OF APPS USED BY SMARTTHINGS USERS.

# of Apps	<5	$5-10$	>10
% of SmartThings Users	48%	43%	9%

TABLE V
OF OPERATIONS THAT FALL INTO EACH RISK CATEGORY & THE RISK RANGE BASED ON THE K-MEAN CLUSTERING. MAKING USERS CONSIDER THE RISK OF AN OPERATION MAKES THEM MORE CONSERVATIVE THAN DOMAIN EXPERTS REGARDING SAFETY OF AN OPERATION.

Risk	Low	Medium	High
Uninformed User	19 (1-1.91)	45 (1.92-2.42)	82 (2.43-5)
Informed User	20 (1-1.94)	57 (1.95-2.59)	69 (2.6-5)
Domain Expert	62 (1-1.83)	58 (1.84-2.83)	26 (2.84-5)

functional-grouping permission model, and then compare that set of accessible operations to a set we obtain after converting the apps to use our risk-based model. In general, we observe that Tyche decreases applications access to risky operations by 60% across all applications. Table VI shows the result of the conversion for our three case-studies.

LockDown locks doors after a specified time of day. To perform its functionality, this application only needs access to "lock()" function. The current permission model, however, gives it access to the "unlock()" function as well. If this application is compromised, an attacker can arbitrarily unlock the doors to the home. Under our risk-based model, the application only asks for low-risk access to the lock, that only grants the "lock()" operation to the app.

SmokeProtector application sounds a siren if the smoke sensor is triggered. Under the current permission model, this application can access the "off()" operation. If compromised, an attacker can turn off the siren. Under our risk-based model, the application only asks for medium-risk access to the speaker. This results in only the "siren()" operation being accessible to the app.

EnergySaver automatically turns off devices around the home if energy consumption is above a predefined value. Under the current model, this application can access the "on()" function of the switches. If compromised, it can turn on switches arbitrarily, thus powering any connected devices. This can negate the purpose of the application or, more dangerously, lead to a fire hazard by powering a device for an extended time, such as a space heater. Under our risk-based model, the "off()" and "on()" operations belong to the same high-risk group. Our users consider "off()" to be high-risk because it can disable safety-critical functions such as life support and refrigeration. Although our model does not separate the "off()" and "on()" operations, at least users are informed of the risks associated with using the appplication. We count the number of high-risk operations that apps can access in the current SmartThings permission model vs. the risk-based model. We find that on average, apps can remain functional and have access to 60% less high-risk operations. Therefore, if an application is compromised, our risk-based model reduces user risk compared to the current permission model.

V. DISCUSSION AND LIMITATIONS

A. Tyche vs. "Permission on first use"

Smartphone platforms generally use either an application install-time permission request modality, or a first-use permission request modality. We surveyed four smart home platforms (SmartThings, AllJoyn, IoTvity, HomeKit), and observed that they currently use install-time permission requests. A potential reason for not including the first-use modality is that it assumes the presence of an interface to the user at all times. Smart home platforms in general might not have traditional user interfaces. For example, Samsung SmartThings only has a hub with no display. In this case, the platform uses an external graphical device, such as a smartphone, to surface the install-time permission request. However, such a device may not always be present at the time of a request. This works well for an install-time modality because the user installs apps with the help of a companion smartphone app. In the case of first-use permission requests, the smartphone may not be present when an application actually accesses a sensitive operation. Tyche permissions can be used with either request modality, however, our prototype currently uses an install-time request.

TABLE VI
WE CONVERTED THREE EXISTING SMARTAPPS TO OUR RISK-BASED PERMISSION MODEL. WE FIND THAT ONLY THE ENERGYSAVER APPLICATION NEEDS HIGH-RISK ACCESS TO A SWITCH. THE OTHER APPS WORK WITHOUT ANY HIGH-RISK ACCESS.

App	Functional-Grouping Model		Risk-Based Model	
	Permissions	**Accessible Operations**	**Permissions**	**Accessible Operations**
Lockdown	lock, contactSensor	**attrs:** lock, contact. **cmds:** lock(), unlock()	lock.medRisk, contactSensor.lowRisk	**attrs:** contact. **cmds:** lock()
SmokeProtector	alarm, smokeDetector	**attrs:** alarm, smoke. **cmds:** off(), strobe(), both(), siren()	alarm.medRisk, smokeDetector.medRisk	**attrs:** alarm, smoke. **cmds:** strobe(), siren(), both()
EnergySaver	powerMeter, switch	**attrs:** power, switch. **cmds:** on(), off()	powerMeter.medRisk, switch.highRisk	**attrs:** power, switch. **cmds:** on(), off()

B. Perception of Risk

A concern with using a risk-based access control system is that different people may have different perceptions of acceptable levels of risk. Although getting locked out may not be a major issue for an adult, it may be problematic for an elderly (or very young) person. Our user study for the prototype was conducted using adults only. Therefore, a limitation is that the resulting risk levels can potentially be biased towards adult perceptions of risk. An interesting future work direction is to examine risk perceptions of different age groups. However, Tyche still provides benefits as it does not make applications more risky than they already are, and it still provides some notion of risk, even if the user's perceptions may not exactly match the system's risk levels.

C. Overprivilege in Tyche

Tyche does not completely eliminate overprivilege but grants applications a risk-based access privilege to each resource. This approach is intuitive—if a user trusts an application with high-risk functionality, they are fine allowing less sensitive functions as well. We observe that assigning risk levels to sensitive operations may or may not change the granularity of the associated permission's operations. This is in part due to our automated clustering process, and in part due to the need to avoid creating many risk levels (*e.g.,* A system with ten or fifteen different risk levels will most likely overwhelm the user and negate any benefits). For some apps (*e.g.,* EnergySaver in §IV-C), this might not result in the app gaining even more limited access to sensitive operations—by a strict definition of overprivilege, even with a risk-based system like Tyche, the app will still be overprivileged. Even in such cases, there are benefits to using Tyche—it serves as a means to notify the user of the riskiness of a particular app.

D. Surfacing Risk Levels to Users

Our user study demonstrates the intuitiveness of a risk-based permission model. Yet, how best to present these risks to the user during installation time remains a challenge. Previous work has looked extensively at this problem in the context of smartphones [7], [8]. Our initial design in Tyche used color coding and a second confirmation for high-risk permissions. We also limited the number of risk levels to three. We leave a user study on the effectiveness of various user interfaces and optimal number of levels as future work. An interesting challenge in surfacing permission requests to users in the context of smart homes concerns the interface mechanism. Hub-based platforms like SmartThings use a secondary device such as a smartphone for surfacing permission requests. Other platforms, such as voice-based assistants like Amazon Alexa or Google Home open up the possibility of a voice-based permission prompt and response mechanism. With a risk-based system, a challenge is to determine how best to leverage these newer modalities to communicate the risk level of an app to the user.

VI. RELATED WORK

Internet of Things platform security is an emerging research area. Fernandes *et al.* [9] performed a security analysis of SmartThings. They showed that more than 55% of SmartThings apps are overprivileged and built four attacks that exploited the overprivilege. Various systems have since studied security issues in IoT platforms. FlowFence [10], [14] proposes data flow protection in IoT frameworks through the use of sandboxes and taint-tracking to impose flow control between data sources and sinks, and is concerned with ensuring that data is not misused. In contrast, risk-based permissions aim to limit the damage a malicious app can cause. ContexIoT is a system that collects contextual information that it uses to help users make more informed decisions when granting permissions to apps [11]. The contextual information is extracted using program analyses techniques. Such contextual prompting can further help in communicating Tyche risk ratings to users. Similarly, Roesner *et al.* introduce Access Control Gadgets (ACGs) and its implementation on Android, LayerCake [16], to improve contextual integrity. Our risk ratings can be made more contexual by adopting ideas from ACGs or from ContexIoT depending on the interface modality that is available to the smart home platform (*e.g.,* classic display, or voice-based

interfaces). However, our goal in this work is to introduce a system design that is aware of risk asymmetries. Similar to ContexIoT, our work adopts a rewriting mechanism to enforce security decisions.

Current smart home platform permissions are modeled after smartphone permissions. There is a large body of work on analyzing and improving smartphone permissions [3]. For instance, Rahmati *et al.* proposed using application context to guide access control decisions [15]. While many of these approaches can be adapted for smart homes, they still overprivilege apps with high-risk operations because they do not take into account risk-asymmetry.

Our user study methodology was inspired by Felt *et al.* [7]. While Felt's study focused on improving the warning messages in smartphone application installation UI, our study focuses on determining the risk level of operations, in order to define various access control levels in each device. Felt *et al.* [8] also performed two user studies on the Android permission model. They found that that only 17% of participants paid attention to permissions during installation and only 3% could answer three permission comprehension questions correctly. These results motivate development of risk-based permission models, where users' understanding of risk factors closely matches domain experts.

More generally, our work draws on results from risk-aware access control systems [5], [13], [6]. Adopting terminology from Petracca *et al.*, we are concerned with three type of risks: (1) risk due to authorizing unsafe operations; (2) risk due to abuse of authorized permissions; (3) risk due to granularity of authorization hooks. Our insight is that current permission models in smart homes do not adequately communicate risk, resulting in users possibly authorizing unsafe operations simply because they do not know the risks involved. The second type of risk occurs due to the modern app model—apps can be malicious, or they can be compromised. The third type of risk occurs due to functional grouping of permissions. Tyche tackles the first and third types of risk creating risk-similar groups of operations, and the communicating those risks to the user. We also introduce a human subjects methodology to estimate risk. However, Tyche limits the effects of the second type of risk, instead of removing it altogether.

Tyche currently uses static estimates of risk. However, risk can change depending on the user. We envision that results from fuzzy MLS systems could be applicable in encoding dynamic risk perceptions into a risk-based access control system [6] for smart homes. We leave this to future work.

VII. Conclusion

Smart home platforms currently use permission models inspired by smartphone OSes. These permission models group device operations based on functionality, and do not take risk-asymmetry into account. Although grouping operations together makes it easier for developers and users to work with permissions, it also overprilvileges apps. Due to the risk-asymmetry, overprivilege can drastically increase the potential for damage if apps are malicious or exploitable. Therefore, we introduced risk-based permissions as an alternative way to group device permissions. We designed Tyche using app rewriting techniques to enforce risk-based permissions, and we conducted a study involving domain experts and Mechanical Turkers to compute user-perceived risk. Our study finds that experts and users share a similar perception of risk. Based on these findings, we re-grouped a physical device's operations into three groups—low-, medium-, and high-risk. We constructed such groupings for 146 operations across 61 types of devices. We evaluated our risk-based model on three existing SmartApps. We found that these apps can be written in a way that reduces access to high-risk operations by 60%.

Acknowledgements. We thank the reviewers for their insightful feedback. This work was supported in part by NSF Grant No. 1646392 and 1740897, the MacArthur Foundation and the University of Washington Tech Policy Lab.

References

[1] *Google Home*. Last Accessed:May 2017.

[2] *Samsung SmartThings Home Automation*. Last Accessed: Oct 2017.

[3] Acar, Y., Backes, M., Bugiel, S., Fahl, S., McDaniel, P., and Smith, M. SoK: Lessons Learned from Android Security Research for Appified Software Platforms. In *IEEE Symposium on Security & Privacy* (2016).

[4] Apple. *HomeKit*. Last Accessed: Oct 2017.

[5] Chen, L., and Crampton, J. *Risk-Aware Role-Based Access Control*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 140–156.

[6] Cheng, P.-C., Rohatgi, P., Keser, C., Karger, P. A., Wagner, G. M., and Reninger, A. S. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2007), SP '07, IEEE Computer Society, pp. 222–230.

[7] Felt, A. P., Egelman, S., and Wagner, D. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices* (2012), ACM, pp. 33–44.

[8] Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., and Wagner, D. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (2012), ACM, p. 3.

[9] Fernandes, E., Jung, J., and Prakash, A. Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy* (May 2016).

[10] Fernandes, E., Paupore, J., Rahmati, A., Simionato, D., Conti, M., and Prakash, A. Flowfence: Practical data protection for emerging iot application frameworks. In *USENIX Security* (2016).

[11] Jia, Y. J., Chen, Q. A., Wang, S., Rahmati, A., Fernandes, E., Mao, Z. M., and Prakash, A. ContexIoT: Towards providing contextual integrity to appified iot platforms. In *Network and Distributed System Security Symposium (NDSS'17)* (2017).

[12] Kim, S.-H., Yun, H., and Yi, J. S. How to filter out random clickers in a crowdsourcing-based study? In *Proceedings of the 2012 BELIV Workshop: Beyond Time and Errors-Novel Evaluation Methods for Visualization* (2012), ACM, p. 15.

[13] Petracca, G., Capobianco, F., Skalka, C., and Jaeger, T. On risk in access control enforcement. In *Proceedings of the 22Nd ACM on Symposium on Access Control Models and Technologies* (New York, NY, USA, 2017), SACMAT '17 Abstracts, ACM, pp. 31–42.

[14] Rahmati, A., Fernandes, E., and Prakash, A. Applying the opacified computation model to enforce information flow policies in iot applications. In *IEEE Cyber-Security Development Conference* (2016).

[15] Rahmati, A., and Madhyastha, H. V. Context-specific access control: Conforming permissions with user expectations. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices* (2015), ACM, pp. 75–80.

[16] Roesner, F., and Kohno, T. Securing embedded user interfaces: Android and beyond. In *Proceedings of the 22Nd USENIX Conference on Security* (Berkeley, CA, USA, 2013), SEC'13, USENIX Association.