Who can Access What, and When?

Understanding Minimal Access Requirements of Building Applications

Jason Koh, Dezhi Hong, Shreyas Nagare, Sudershan Boovaraghavan, Yuvraj Agarwal, Rajesh Gupta
{jbkoh,dehong}@ucsd.edu,{nagare,sudershan,yuvrajagarwal}@cmu.edu,rgupta@ucsd.edu
University of California, San Diego, Carnegie Mellon University

ABSTRACT

Smart building applications (apps) are faced with the real challenge of unfettered access to mission-critical building resources that makes buildings vulnerable to attacks and occupants to privacy invasions. Existing methods that group users for access control are too coarse-grained to avoid granting over-privileges to apps. Furthermore, they lack means to model, express, and use access patterns that can be critical in securing automated building operations. In this paper, We identify Who, What, and When as the key information dimensions for building apps access control after thoroughly reviewing 125 smart building app publications in two major venues. Our analysis reveals that dynamic access control requires unique access patterns of individual apps, as well as the building and user context. We also observe that existing Building Operating Systems and IoT platforms fall short of sufficiently representing all the necessary patterns, and further discuss future directions for the design of access control systems needed to support building apps.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; • **Computer systems organization → Sensor networks**.

KEYWORDS

Access control, smart buildings, IoT

ACM Reference Format:
Jason Koh, Dezhi Hong, Shreyas Nagare, Sudershan Boovaraghavan, Yuvraj Agarwal, Rajesh Gupta. 2019. Who can Access What, and When?: Understanding Minimal Access Requirements of Building Applications. In *The 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '19), November 13–14, 2019, New York, NY, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3360322.3360868

1 INTRODUCTION

Buildings as a common application (app) platform have great potential to save energy and improve the quality of life for the occupants. Building apps are connected to buildings through Building Operating Systems (BOS) [2, 7] to read sensors and control actuators connected to the resources in the buildings. Similar to the challenges with overprivileged apps on smartphones [6], smart building apps become vulnerable attack surfaces when they are granted too much access to the underlying resources. To minimize the risk of a security breach, *Access Control* is a must: Through the BOS, a manager of the building should decide whether an app can read or control the requested resources, in order to grant the app access to only the necessary resources for its functionality.

However, unlike smartphones where users control their own resources (e.g., camera), the smart building platform is a multi-tenant environment. In such a shared platform, multiple users may access the same resources via various apps, and the users usually are not the resource managers. For example, if occupants want to remotely control the Heating, Ventilation, and Air Conditioning (HVAC) through a web interface, a building manager is actually the one in charge of operating the equipment and can approve such control. Furthermore, an occupant may have different purposes for different apps, e.g., a lighting control vs the HVAC app, with different access permissions.

Due to the multi-tenancy, apps are frequently overprivileged in existing infrastructure and subject to data exfiltration. For example, Genie [5] is a software thermostat through which registered occupants control the HVAC in their offices, built on top of the BuildingDepot [2] (BD) BOS. However, since user management is delegated to Genie and not BD, BD provides the Genie app unfettered access to all the HVAC terminal units in the building, even though only 15 rooms are actively controlled by Genie every day.

Traditional access control patterns fail for two main reasons for buildings. Access Control Lists (ACL) define fine-grained permissions to control what actions each user of an app can take on each object. Using ACLs, a resource manager would need to manually approve the access per user per app, which quickly becomes untenable. Grouping resources and users may provide more descriptive patterns for access control. However, existing patterns, such as attributes [8] and roles [10], are not specific enough to represent only the exact resources in buildings. Access control patterns for building apps should be specific enough for resource managers to balance expressiveness, understanding, and scalability.

In this paper, we review 125 papers on building apps at two major venues in the last seven years to extract common resource access patterns. We identify six high-level and 20 specific app categories. We also find that `Who`, `What`, and `When`, are the three authoritative dimensions for determining users' access to the resources. We show that these information dimensions are largely neglected in existing BOSes and IoT platforms, resulting in an unmanageable app approval process and possibly overprivileged apps. Based on our observations, we also provide design suggestions for access control to BOSes for wider adoption of building applications.

BuildSys '19, November 13–14, 2019, New York, NY, USA

ACM ISBN 978-1-4503-7005-9/19/11…$15.00
https://doi.org/10.1145/3360322.3360868

2 PROBLEM FORMULATION

In the context of buildings, *resources* refer to the entities essential to building apps [4], including all the physical points (e.g., sensors), equipment (e.g., HVAC), the data generated by these points, and the physical space (e.g., offices). We assume all the resources are managed through a Building Operating System (BOS), which provides programming interfaces for apps to interact with the resources in the building. We thus regard a BOS as a trusted information source so that we can augment its security measures on demand.

An app can manifest the necessary resources and the *owners* should approve the app's access. As a building is typically managed by a building manager on behalf of the owner, (s)he has the authority to arbitrate actions over the resources. However, a third-party company may also deploy and manage additional resources in a building, and could directly manage these resources or delegate the control to the building manager. In other words, multiple resource owners might need to work together to decide on the access policy.

The various *users* (e.g., occupants, building managers, etc.) would inevitably have different demands, and thus should be granted different permissions. For example, an occupant should be able to change the temperature in her office but not her colleagues'. Since different apps and the users need to access different resources, and there could be hundreds of apps with thousands of users, to manually create and maintain access control lists would require much effort from the corresponding resource owners. Thus, the capability to automatically represent and check what users can do with apps is crucial to scaling access control for building apps.

3 RESOURCE ACCESS PATTERNS OF APPS

3.1 Analysis Setup

To holistically understand different access patterns, we have reviewed 125 papers published at BuildSys from 2009 to 2018 and e-Energy from 2012 to 2018. The authors of these papers have diverse industry and academic backgrounds, use heterogeneous testbeds ranging from residential to office buildings, and cover an extensive range of building apps. While Balaji et al. identified eight app categories [4], we expand them to 20 categories under six high-level domains[1]. The first and the second columns of Table 1 list the app domains and categories, respectively. The authors reviewed the papers and at least two authors cross-validated the results of each paper. The full analysis is available online[2], and we cite app papers with their row number in the this document as (R#). We identify three information dimensions that should be examined for building apps to access the required resources:

- **Who** may use this app?
- **What** are the resources this app can possibly access?
- **When**, or in which context, can a user use this app?

Table 1 shows the access patterns required for each app category, based on the three dimensions. They are complete in expressing the representative apps studied in this paper and we expect them to generalize to other building apps.

3.2 Who: User Type

We mainly identify five types of users within buildings. **Occupants** typically use apps to control their environment (e.g. manage temperature) or understand their behaviors (e.g., energy usage). **Building Managers** oversee the operation of building with regard to space management, equipment maintenance, energy efficiency, maintaining security, etc. They may use most of the apps except the apps that primarily produce indirect results such as Energy Models. Some apps are designed to be used by **Other Apps**: Analytical apps, such as a prediction model of an electrical device's energy consumption, may feed their output to other operative apps such as demand response. **Energy Providers** are a unique type of users who do not reside in target buildings. They collect information about a building's energy usage and use that to control the building equipment either directly or indirectly through utility pricing or demand response events, in order to stabilize the electrical grids. Some apps are agnostic to user types, e.g., a public energy dashboard (R9) or a location-based controller (R60) , which **Anybody** can use.

3.3 What: Resource Identification

Resource Types: Resource type is the most important class of information, as any building app must access some resource(s), whether it be a temperature sensor, a light bulb, or an office. The resource type is also critical for reasoning about security since different resource types have different capabilities, with different consequences if breached. For example, upon a security breach, a motion sensor may leak private occupancy information while an airflow setpoint could physically damage the controlled equipment.
Resource Relations: Relations between resources connote their relative functionalities, enabling precise resource identification. For example, the causal control dependence between points are critical for Fault Detection and Diagnostics (FDD) (R33) ; when the supply airflow of a VAV is anomalous, the corresponding actuator in the same VAV are required for analysis. Except for apps that need to access all the resources of some type (e.g., a tool visualizing all the building energy meters (R9)), resource relations are a crucial information dimension for describing apps' requirements.
Resource-User Relations: Resources serve, monitor, and/or are controlled by users, and thus expressing these relationships is a key. We identify three kinds of such relations. First, in buildings, **space** is often assigned to a person whether it is an office, a lab, or a desk. Consequently, a sensor in an assigned space reveals information about the person, e.g., presence or schedule. Thus, space-related apps, such as occupancy detection, need to get approval from the associated person who is being monitored when accessing these resources. Note that while theoretically everyone needs to approve an app's access to their data, this process may be delegated to someone like a building manager. Delegation is out of the scope of this paper though it is complementary to any access conditions. Second, people use personal or allocated **devices** to customize the indoor environment such as lighting (R60) . It is thus necessary to consider *what devices each user has*. Lastly, user **preferences** are frequently used in apps that improve the occupants' comfort and productivity. We discover that multiple apps need user preferences over conditions such as lighting and temperature to control the environment for the users based on their locations (R43) .

[1] The venues lack some app categories such as building security and healthcare though our analysis applies to undiscovered categories in a similar manner.
[2] https://tinyurl.com/building-apps-access-control

Table 1: Resource Access Patterns across Different App Categories. An app may access only the resources meeting all the marked conditions. "rsrc" stands for resource and "↔" for a relation between the two classes. Brick [4] comprehensively models "rsrc type" and "rsrc↔rsrc" but not the rest of the table.

			Who					What					When					
App Domain	App Category	#Apps	occupant	building manager	other apps	energy provider	anybody	rsrc type	rsrc↔ rsrc	user↔ space	user↔ device	user↔ preference	user location	rsrc state	calendar schedule	dr event	onetime access	user request
Maintenance	FDD	6	◔	●	○	○	○	●	●	○	○	○	○	○	○	○	◑	○
	Space Management	2	○	●	◕	○	○	●	●	◕	○	○	○	○	◕	○	◕	○
Building Modeling	Environment Model	2	○	●	○	○	○	●	●	○	○	○	○	○	○	○	○	○
	Structural Model	1	○	●	●	○	○	●	●	○	○	○	○	○	○	○	●	○
Energy Analysis	Energy Disaggregation	20	◕	●	◕	◔	○	●	◕	◔	◔	○	◔	◔	○	○	◑	○
	Appliance Identification	1	○	●	●	○	○	●	●	○	○	○	○	○	○	○	●	○
	Thermal Comfort Model	7	◕	◑	◔	○	○	●	◕	◑	○	◔	◑	○	○	○	◔	◔
	Energy Model	8	◔	●	●	◕	○	●	●	○	◔	○	○	○	○	○	◕	○
	Energy Footprinting	5	●	●	◕	○	○	●	◑	◑	◔	○	○	○	○	○	○	○
Efficient Control	Model-Predictive Control	11	◔	●	◔	○	○	●	●	○	○	○	○	○	○	◔	◔	○
	Occupancy-Based Control	21	◑	●	◔	◔	○	●	◕	◑	◔	◔	◔	◔	◔	○	◔	○
	Demand Response	20	◑	●	○	◕	○	●	◕	○	◔	◔	○	○	◔	◕	◔	○
Occupancy Modeling	Occupancy Detection	13	◔	◕	●	○	○	●	◕	◔	◔	○	○	○	◔	○	◑	○
	Occupancy Identification	4	◕	●	◕	○	○	●	◕	◑	○	○	○	○	○	○	◑	○
	Activity Recognition	3	◕	●	◕	◑	○	●	●	◑	◕	○	◑	○	○	○	◑	○
	Behavior Modeling	2	○	●	●	○	○	●	◕	○	●	○	○	○	○	○	○	○
User Interface	Web Displays	2	●	◕	○	○	◕	●	◕	○	○	○	○	○	○	○	○	○
	Remote Controller	2	●	○	○	○	●	●	●	○	○	○	○	○	○	○	○	●
	Participatory Sensing	5	●	◔	○	○	○	●	◕	◑	○	○	◑	○	○	○	◔	◔

●: #apps > 75% ◕: 75% ≥ #apps > 50% ◑: 50% ≥ #apps > 25% ◔: 25% ≥ #apps > 0% ○: no apps

3.4 When: Access Context

Resource access may be temporally granted based on the user/app context, so as to prevent overprivileged apps with constant access.
User Location: As a user is physically present in a particular space, apps related to occupants can refer to the user's location. For example, an app for occupancy-based HVAC control (R72) is currently granted access to all the VAVs in an entire building all the time, whereas it should access only some VAV when the user is nearby.
Resource State: An app may need to be active based on resources' states. For example, a lighting controller should be active only when the associated room is occupied. Note that, the difference between *user location* and *resource state* is that the user's location is specific to a target user while a resource's state is occupant agnostic.
Schedule: Temporal bounds may be explicitly defined, whether they are regular schedules or temporal bookings.
Demand Response: Demand Response (DR) events rarely occur – usually several times per year even for program participants. Only when a DR event happens should automated DR apps be active, which have powerful capability to control the entire buildings.
One-time Access: Data-driven apps need to access historical data for training, but once they train their models, the data should not be accessible and only the trained models should be used.
User Request: Apps such as remote controllers convey users' intention to control the system. An app's request should be valid only when it can be verified to be from the actual user.

3.5 An Example for Access Pattern Evaluation

To evaluate and approve an access request from an app, the BOS needs to first identify the app and user, i.e., authentication, and then check its *access pattern* — whether the request satisfies the information dimensions (i.e., columns defined in Table 1), including the user's role, whether the user is trying to access permitted resource(s), whether the user's demand has expired, etc. For example, an HVAC remote control app can be authorized only when an *occupant* (user type) is trying to control the *temperature setpoint* (resource type) *of the terminal unit* (resource-resource relations) in *his/her office* (resource-user relation) when (s)he *sends a request* (user request event).

However, we shall note that, as each app may uniquely describe its required resources and the relations among them, i.e., resource group, the request approval process thus involves interpreting a potentially tremendous set of combinations. Since it is impossible to exhaustively predefine static resource groups in BOSes to cover all the possible patterns, and rather, BOSes should be able to verify different information at runtime.

4 EXISTING ACCESS CONTROL PLATFORMS

BOSes (and IoT platforms) use different access control models for evaluating information sources, as summarized in Table 2.

General IoT cloud services (e.g., AWS IoT) and commercial BOSes (e.g., NiagaraAX [12]) support only ACLs and roles [10] for authorization.An ACL is a list of permissions for various users on a single

Table 2: Access control patterns supported in existing Smart Building and IoT platforms.

System	Who	What	When
Wave [3]	Individuals	Predefined groups	No
BuildingDepot3 [2]	Custom groups	Tag-based groups	No
NiagaraAX [12]	Individuals & Roles	Each object	No
Cloud IoTs [1]	Individuals & Roles	Each object	No
ESO [11]	Individuals	Each object	Yes

resource and each resource maintains its unique ACL. Thus, although ACLs and user roles alone could provide stringent access controls, even in one building, a system manager would have to laboriously maintain an immense number of ACLs for tens of apps and hundreds of users. To overcome ACLs' limited expressibility, many access control patterns have been proposed for multi-tenant cloud platforms [9]. However, these models are often too specialized for computational resources, lacking interleaved relationships, and the context barely changes over time while the users' behaviors and built environments change over time.

Wave [3] and BuildingDepot3 (BD3) [2] support grouping resources for authorizing apps. Wave allows delegating the authorization of a group of resources to another entity, but does not specify the definition of groups, which could be based on the hierarchy of location or equipment. In addition, apps need different groupings to follow the principle of least privilege, as they may need different resources in the same group (e.g., on the same floor.) For example, assuming an instance of the HVAC remote control app implemented with Wave that groups the resources based on rooms, it will be allowed to take any actions on all the data points (commonly ~15) in a room, whereas it only needs to read the temperature sensor and change the temperature setpoint. Such grouping unnecessarily exposes more resources to the app and the ability to control safety-related points such as a minimum airflow setpoint.

BD3 [2] allows flexible grouping over resources based on tags. For example, a group may consist of all the sensors with the "temperature" tag. BD3's grouping is more flexible than Wave's as a resource may belong to multiple groups. Still, the groups are manually defined and might not cover all the possible patterns.

Furthermore, Wave and BD3 do not employ contextual information such as a "user request" event. An occupant might use the HVAC remote controller infrequently, e.g., when she is working outside the regular schedule or during atypical weather. Thus, the app does not have to be granted access for the user all the time. Instead, a BOS should incorporate a temporal authorization, such as to set an expiration date or to schedule timed activation, while tracking the relevant events from trusted sources. Environmental Situation Oracles (ESOs) [11] perform access control based on events that can be generalized into When-type information. It is necessary to incorporate dynamic events with metadata from different sources for the most general and expressive access control patterns.

Still, while buildings are a multi-tenant platform, none of the aforementioned systems are designed for apps serving multiple users, but rather they consider an app as a standalone entityWhen the HVAC remote controller is implemented with these systems, the app would have permissions for the superset of all the possible users' all the time, thus being unnecessarily overprivileged. In other words, the app would be able to access hundreds of the terminals units in the entire building all the time, while it should only access around 10 rooms' terminal units per day on average.

5 DESIGN SUGGESTIONS

We have shown that Who, What, and When are the three authoritative information dimensions to tightly describe all the apps' access patterns we studied, which none of the existing BOSes expose effectively. For the secure adoption of smart building apps at scale, we recommend several design considerations for access control:

1. Instead of preconfiguring resource groups, BOSes should adopt flexible groupings over the complete view of building systems due to the heterogeneous apps' access patterns. (What)
2. BOSes should have a way to join a building's metadata with the users', and users should be a part of the system modeling process. (Who and What)
3. The metadata of buildings and users should be rigorously maintained. Access control patterns would rely on the metadata, applied to all the resources. Automated verification processes for metadata correctness is also desirable.
4. Resources' context should be easily verifiable and it should be describable inside access patterns from apps (When).
5. An app's access capability should be dynamically determined based on its users and the context at runtime, instead of subsuming all the potential access requests.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation [1526841, 1526237, 1564009, 1801472].

REFERENCES

[1] 2019. AWS IoT Authorization. https://docs.aws.amazon.com/iot/latest/developerguide/authorization.html. (2019). accessed 2019-06-22.

[2] 2019. BuildingDepot 3.0. https://buildingdepot.org/. (2019). accessed 2019-06-22.

[3] Michael P. Andersen, Sam Kumar, Moustafa AbdelBaky, Gabe Fierro, John Kolb, Hyung-Sin Kim, David E. Culler, and Raluca Ada Popa. 2019. WAVE: A Decentralized Authorization Framework with Transitive Delegation. In *28th USENIX Security Symposium*.

[4] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, et al. 2018. Brick: Metadata schema for portable smart building applications. *Applied energy* 226 (2018), 1273–1292.

[5] Bharathan Balaji, Jason Koh, Nadir Weibel, and Yuvraj Agarwal. 2016. Genie: a longitudinal study comparing physical and software thermostats in office buildings. In *UbiComp*.

[6] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I Hong, and Yuvraj Agarwal. 2017. Does this app really need my location?: Context-aware privacy management for smartphones. *IMWUT* 1, 3 (2017), 42.

[7] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. 2013. {BOSS}: Building operating system services. In *NSDI*. 443–457.

[8] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. 2013. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication* 800, 162 (2013).

[9] Canh Ngo, Yuri Demchenko, and Cees de Laat. 2016. Multi-tenant attribute-based access control for cloud infrastructure services. *Journal of Information Security and Applications* 27 (2016), 65–84.

[10] Ravi S Sandhu. 1998. Role-based access control. In *Advances in computers*. Vol. 46. Elsevier, 237–286.

[11] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2018. Situational Access Control in the Internet of Things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1056–1073.

[12] Tridium. 2017. Niagara Enterprise Security. https://tinyurl.com/yyh5q6ek. (2017). accessed 2019-06-22.