Effectively Enforcing Authorization Constraints for Emerging Space-Sensitive Technologies

Carlos E. Rubio-Medrano
Arizona State University
Tempe, Arizona, USA
crubiome@asu.edu

Shaishavkumar Jogani
Arizona State University
Tempe, Arizona, USA
sjogani@asu.edu

Maria Leitner
AIT Austrian Institute of Technology
Vienna, Austria
maria.leitner@ait.ac.at

Ziming Zhao
Rochester Institute of Technology
Henrietta, New York, USA
ziming.zhao@rit.edu

Gail-Joon Ahn
Arizona State University and
Samsung Research
gahn@asu.edu

ABSTRACT

Recently, applications that deliver customized content to end-users, e.g., digital objects on top of a video stream, depending on information such as their current physical location, usage patterns, personal data, etc., have become extremely popular. Despite their promising future, some concerns still exist with respect to the proper use of such *space-sensitive applications* (S-Apps) inside independently-run physical spaces, e.g., schools, museums, hospitals, memorials, etc. Based on the idea that innovative technologies should be paired with novel (and effective) security measures, this paper proposes *space-sensitive access control* (*SSAC*), an approach for restricting space-sensitive functionality in such independently-run physical spaces, allowing for the specification, evaluation and enforcement of rich and flexible authorization policies, which, besides meeting the specific needs for S-Apps, are also intended to avoid the need for interruptions in their normal use as well as repetitive policy updates, thus providing a convenient solution for both policy makers and end-users. We present a theoretical model, a *proof-of-concept* S-App, and a supporting API framework, which facilitate the policy crafting, storage, retrieval and evaluation processes, as well as the enforcement of authorization decisions. In addition, we present a performance case study depicting our *proof-of-concept* S-App in a set of realistic scenarios, as well as a user study which resulted in 90% of participants being able to understand and write authorization policies using our approach, and 93% of them also recognizing the need for restricting functionality in the context of emerging space-sensitive technologies, thus providing evidence that encourages the adoption of *SSAC* in practice.

CCS CONCEPTS

• **Security and privacy** → **Access control**; • **Human-centered computing** → *Mixed / augmented reality*;

KEYWORDS

Attributes; Authorization Policies; Space-Sensitive Access Control

ACM Reference Format:
Carlos E. Rubio-Medrano, Shaishavkumar Jogani, Maria Leitner, Ziming Zhao, and Gail-Joon Ahn. 2019. Effectively Enforcing Authorization Constraints for Emerging Space-Sensitive Technologies. In *The 24th ACM Symposium on Access Control Models and Technologies (SACMAT '19), June 3–6, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3322431.3325109

SACMAT '19, June 3–6, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6753-0/19/06…$15.00
https://doi.org/10.1145/3322431.3325109

1 INTRODUCTION

Computing is nowadays transitioning from rigid stationary locations, e.g., warehouses, racks and desktops, to a so-called state of *ubiquity*, where it can be exercised in many different physical spaces thanks to recent advances in mobile hardware devices, location-based services such as *global positioning systems* (GPS), and dedicated support from operating systems and hardware-control drivers. In this context, a plethora of software applications have been recently developed to provide highly-customized content based on input information they actively collect behind the scenes from end-users, e.g., physical location, personal info, usage patterns, etc., allowing for an enhanced experience and a closer mode of interaction. Such applications, hereafter to be referred as *space-sensitive apps* (S-Apps), depict an emerging trend that is likely to stay at the present and future of all computing applications. Exemplary S-Apps currently available in the market include, but are not limited to the following: contemporary mapping apps such as Google Maps [11], *online social networks* (OSNs) [9], *mobile augmented reality apps* (MAR) [5] such as Pokémon GO [17], and even the recently-introduced applications renting electric-based scooters [15], which have become increasingly popular in many cities worldwide. All of these S-Apps display customized functionality that varies significantly depending on location-based information, and they are produced/supported by different, independently-run companies, ultimately providing an enhanced experience that encourages end-users to explore their physical surroundings in an active way.

Despite their recent success, some issues still remain concerning the proper use of S-Apps inside independently-run physical spaces that may be under the actual control of third-parties other than the end-users themselves. As an example, concerns have been

raised about users playing S-Apps in places such as a WWII Museum in Germany [6], a park in Australia [4], as well as a 9/11 memorial in the United States [23]. Therefore, there is a need to restrict space-sensitive functionality in physical spaces for different applications, users, devices, etc. Concretely, *space owners*, e.g., government agencies, homeowners, renters, etc., should be allowed to restrict the functionality of any S-Apps in the spaces under their control. Moreover, *application developers*, either individuals or organizations, may also want to restrict the functionality of their S-Apps in some spaces, e.g., companies restricting their apps for certain markets only such as North-America or Central Europe.

However, restricting space-sensitive functionality over physical spaces impose a variety of interesting challenges: first, there is a need to support the specification of a wide range of space-sensitive functionality, which may include, but may not be limited to the following: displaying digitally-created objects on top of a video stream, playing customized audio files, activating the electric motors of a scooter, etc. Second, runtime information obtained from different actors, namely, the S-Apps themselves, developers, mobile devices, device manufacturers, and end-users, should be also leveraged when deciding functionality restrictions. Third, there is also a need to minimize updates on existing or new restrictions, e.g., space owners should not be required to update their preferences every single time a new S-App hits the market. Fourth, end-user interaction, e.g., managing permissions when approaching a given space, should be also minimized, in an effort to prevent the continuous disturbance of S-Apps that can result in frustration and their consequent disregard by end-users. Finally, an approach for restricting space-sensitive functionality must allow for both end-users and developers to clearly and unambiguously specify restrictions at the same time implementations at the source-code level are facilitated.

In order to solve these problems, this paper presents *space-sensitive access control* (*SSAC*), a novel approach that introduces a new dimension in authorization in order to restrict the space-sensitive functionality offered by S-Apps in the context of multiple, independently-run physical spaces. Our approach is based on precisely modeling security-relevant information in the form of *attributes* [12], an emerging paradigm that can support a range variety of S-Apps, physical locations, mobile devices, as well as abstract concepts such as time, thus effectively supporting the specification, evaluation and enforcement of rich and flexible *one-size-fits-all* authorization policies. In addition, our *SSAC* is intended to stay manageable for end-users, policymakers and developers without the need of extensive and comprehensive training in authorization topics, thus enhancing its suitability for being deployed in existing and future S-Apps.

With all this in mind, this paper makes the following contributions: first, we elaborate on the unique problems and challenges introduced by emerging S-Apps and similar technologies aimed at providing ubiquitous computing in the context of independently-run physical spaces. Second, we provide *SSAC*, a novel authorization approach that allows for S-Apps to become *policy-governed* when it comes to render space-sensitive functionality, at the same time it stays manageable for policy makers, end-users and developers. Third, we introduce a set of metrics for evaluating the runtime performance of approaches tailored for restricting space-sensitive functionality by means of authorization policies, which we then use as a part of a series of a case study evaluating our *SSAC* approach. In addition, we also present the rationale behind and the results of a user study focused on the ability of end-users to understand and write authorization policies using *SSAC*, which combined provide evidence of the suitability of our proposed approach to be fully deployed in practice.

2 RELATED WORK

2.1 Emerging Space-Sensitive Technologies

Research on security and privacy in the context of emerging space-sensitive technologies has been growing recently. Roesner et al. [21] proposed an approach that allows for real-world objects to specify access control policies in the context of MAR. A trusted security module senses these policies and adapts the views (video streams) accordingly, e.g., stopping video recording in bathrooms and removing bystanders from videos, introducing the concept of *passports* to support policy authenticity (QR codes). While this approach focuses on the sensing of policies when mobile devices are already up and running, it can be difficult to provide passports in large or complex locations (stadiums, concert halls). Also, this approach heavily depends on the video-based recognition performance of the device. A similar approach by Jana et al. [13], introduced dedicated placeholders known as *recognizers*, allowing for S-Apps to be granted specific permissions specified by end-users, thus ultimately restricting the space-sensitive functionality by means of operating system calls. While serving as an inspiration, these approaches do not handle the collection of security-relevant information for the purposes of policy specification and evaluation, which allows for extended flexibility and convenience, as our approach does. In addition, our solution can be extended to additional S-App technologies other than MAR and does not require any visible passports and centers to be placed in physical locations.

2.2 Location-based Access Control

Location-based access control (LBAC) is mostly concerned with making access control decisions using the current physical position of end-users [8]. As an example, a doctor may be able to access patient records when he/she is on the premise of a given hospital. Over the years, many location-aware access control models have been developed. Ardagna et al. [1, 2] described an authorization model supporting location-based conditions. However, such location-based conditions are imposed on the location of the end-user itself but not on a protected space as defined by a policy maker. Furthermore, location-aware models for *role-based access control* (RBAC) [22] have been widely discussed in the literature. For example, the GEO-RBAC model proposed by Damiani et al. [7] enhances the RBAC model with spatial and location-based information, allowing for spatial entities to model geographically bounded roles. In a similar approach, Miettinen et al. [16] presented an approach for automatically detecting the context a given user may be in, based on information obtained from different sources, including location as well as the set of other users that may be in the surroundings, in an attempt to automatically tailor authorization policies based on the detected context. While extremely convenient, the

approaches just discussed have mostly focused on single-managed spaces, which may support a small set of authorization policies (mostly a single organizational one), and whose underlying authorization model may not natively support the use of multiple pieces of runtime information for making access decisions. Moreover, previous approaches also assume a well-trained cybersecurity official may serve as a policymaker and may also supervise the deployment of the authorization enforcement mechanisms and auxiliary systems. As it will be further discussed in Sec. 3, such an assumption may not necessarily hold in the landscape of S-Apps.

3 PROBLEM STATEMENT

As introduced in Sec. 1, despite their recent success, functionality of S-Apps has been deemed as inappropriate in different cases observed in practice. As a running example, graphically shown in Fig. 1, consider a sample gaming S-App, similar to the ones that have been the subject of concerns in practice [6] [4] [23]. With that in mind, there is a need for an approach that can effectively describe space-sensitive functionality, e.g., digital objects, the different operations that can be performed over it, e.g., rendering over a video stream, as well as any other security-relevant information that becomes relevant within the context of a protected space, e.g., information about the S-Apps themselves such as names, categories, time, etc.

Protected Spaces. A protected space can be defined as a physical space that can be either two dimensional such as plain surfaces over terrain, or three dimensional such as building floors. In addition, protected spaces can be identified by means of GPS coordinates, addresses, sites/markers (parks, schools, museums, etc.) or by means of spatial ranges, (cities, states and countries).

Space Ownership. *Space owners*, e.g., homeowners, renters, government agencies, institutions, etc., can be also unambiguously related to a given space by an authority entity who can securely verify their identities at the city, state or nation-wide level, e.g., by implementing a trusted verification/certification mechanism. For the purposes of this paper, we assume the process of space ownership has been successfully performed beforehand, and therefore, is deemed as out of scope.

Policy Makers. In addition, we also define two classes of policy makers: first, *space owners*, which may want to restrict the functionality of space-sensitive S-Apps under the physical spaces they control. Second, *application developers*. which may also want to restrict the functionality of the space-sensitive S-Apps they produce in a predefined set of spaces.

End-Users. Similarly, end-users are the human beings in control of S-Apps who are ultimately the target of customized space-sensitive functionality while entering/leaving/staying at protected spaces. In some contexts, end-users may also become policy makers themselves, assuming they have ownership rights over a protected space as mentioned before. However, for the purposes of this paper, the sets of policy makers and end-users are not to intersect with each other unless it is explicitly stated.

The Landscape of Emerging S-Apps. As illustrated by the motivating examples described before, S-Apps introduce a set of interesting challenges for providing authorization guarantees for

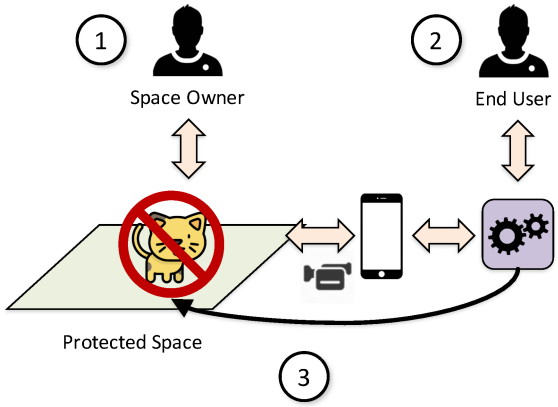


Figure 1: A graphical description of our problem statement: a *space owner* wants to restrict the content that is displayed by S-Apps on a given *protected space* (1). Later, an end-user approaches the protected space with an S-App activated (2). The S-App then displays digital content using the device's built-in video camera (3). *Space owners* should be able to express their preferences with respect to the space-sensitive functionality that can be displayed under the protected spaces they control.

our so-called *protected spaces*. First, there may exist multiple, *heterogeneous*, independently-run, protected physical spaces, each of them possibly implementing their own security protection domain. Second, each of these protected spaces may in turn implement their own set of authorization policies, which may depict a variety (or a combination) of classical authorization models, e.g., RBAC, etc., as well as emerging methodologies such as *attribute-based access control* (ABAC) [12], which may be favored due to their extensive support for the use of multi-source runtime information for convenient and effective access mediation. Third, the authorization domains of most of these protected spaces may be potentially managed by end-users who may not have received proper training in cybersecurity and/or computer science. As an example, small schools willing to restrict the use of gaming S-Apps by technological means (rather than simply issuing explicit prohibitions likely to be disobeyed by students), may not have a dedicated cybersecurity officer in place, but rather a school official that may have received basic training to fulfill such a task, therefore requiring end-users serving as policy makers to write policies protecting their physical spaces. In addition, end-users of S-Apps, e.g., students, may also need to understand policies protecting the spaces they entered to while using S-Apps, and they may also need to protect their sensitive data while requesting for authorization in protected spaces. As an example, students may not want to disclose unnecessary personal information to any authorization enforcement mechanism that is in place in their school. Fourth, the authorization policies in place for protecting physical spaces, as hinted before, may make use of extensive runtime information obtained from a variety of sources for the purposes of better describing the many different access needs of heterogeneous, independently-run organizations and individuals serving as space owners.

Limitations of Existing Attribute-based Models. Despite its emerging popularity, existing ABAC-based paradigms, e.g., the *extensible access control markup language* (XACML) [19], may not well-suited for the requirements we have just described. As an example, XACML, despite being XML-based and therefore tailored for both human and machine-based readability, is considerably complex to be handled by end-users lacking extensive training in authorization: its syntax, semantics, built-in functions, combining algorithms, etc., represent a considerable challenge even for expert policy makers [3], which may ultimately result in the introduction of costly errors and omissions that may open the door for security vulnerabilities and/or incidents. In addition, tool support for XACML and other ABAC approaches is still in its infancy, as support for policy crafting, deployment, and enforcement must be provided from customizing existing APIs, which is not a straightforward task, requiring a considerable organizational and human effort. Finally, there is still no well-accepted description of ABAC in the literature, just like the one other classical models benefit from, e.g., RBAC [22]. As an example, there is still discussion on how to properly represent attributes and model authorization constraints [10], and how to support other classical authorization models with an attribute-based approach [20].

4 OUR APPROACH: SPACE-SENSITIVE ACCESS CONTROL FOR S-APPS

In order to solve the challenges and overcome the limitations just presented, we now present *SSAC*, an effective, easy-to-understand, easy-to-enforce model that 1) it is based on attributes, making it flexible enough so it can be customized to meet the many different authorization needs exhibited by independently-run protected spaces; 2) it is efficient enough to model accurate authorization needs, providing a tradeoff between permissive and strict policies at the same time it can be enforced in practice without introducing noticeable performance, e.g., delaying the normal functioning of S-Apps as a result of policy-enforcement routines; 3) it is accessible to end-users, e.g., home owners, building administrators, etc., without extensive training in authorization and/or cybersecurity topics; and, 4) it is convenient for developers of S-Apps, as it provides well-defined foundations and a supporting implementation framework, thus relieving developers from having to implement such a new model from scratch. We start our discussion with a general description of *SSAC*, including different components involved and the relationships between them. Next, we elaborate on the need for different participants within the context of S-Apps to collaborate with each other in order to fully support it. Finally, we round up the discussions introduced in this chapter by providing a well-defined theoretical model.

4.1 Main Components of *SSAC*

Policy-governed S-Apps Using the *SSAC* we will further discuss in this Section, S-Apps can become *policy-governed* by implementing control logic to effectively restrict space-sensitive functionality in the context of a certain protected space. Fig. 2 provides a graphical depiction of our proposed solution: when approaching a protected space, a policy-governed S-App, acting on behalf of the end-user, obtains information about the current physical location, the S-App itself, and even the current end-user, and prepares an authorization request, which is then forwarded to a supporting implementation framework. Policy evaluation results are returned back to the S-App, which implements control logic to provide only authorized functionality, thus being effectively *governed* by the authorizations decisions obtained in the context of a given space.

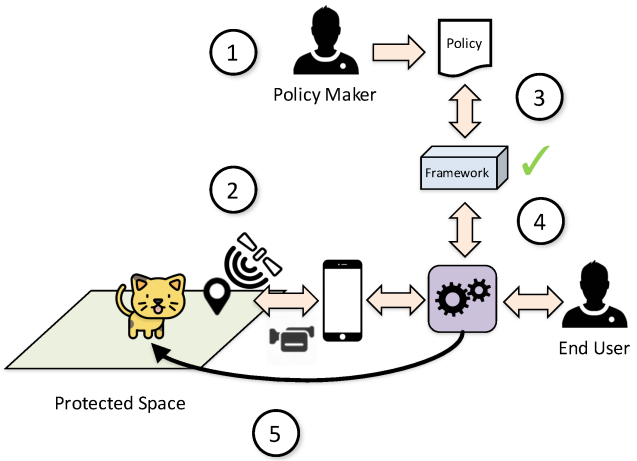


Figure 2: A graphical description of our proposed *policy-governed* S-Apps: a policy maker leverages our framework to craft a policy over a protected space (1). Later, an end-user approaches the protected space with an S-App activated, and the current end-user current location is detected by a sensor (2). The S-App then requests the framework for authorization to provide functionality over the detected protected space (3). Once a response is received, the S-App renders only authorized functionality (4) (5).

Attributes. As stated in Sec. 3, security-relevant information, which may be obtained at runtime, can be leveraged for the purposes of authorization, as the process of authorizing space-sensitive functionality for S-Apps may certainly benefit from modeling such security-related information in a concise and well-defined way. As an example, syntax and semantics can be properly understood by makers when crafting, evaluating and enforcing authorization policies. Lately, *attributes* [12] have been recognized as a convenient way to represent security-relevant information that may be either generated at runtime, e.g., the current time when a given policy is to be evaluated, or may have been also generated beforehand, e.g., a description of student membership issued by a university. In our approach, we represent attributes as a 3-tuple consisting of the following elements: 1) a *datatype*, which, as expected, defines the type of data as well the range of values an attribute instance may take. 2) a *name*, which uniquely identifies the attribute within the context of a given implementation of our approach. 3) the set of *values*, which represents a unique subset of the values defined by the datatype component.

Following our running example introduced in Sec. 3, the overlapping of digital content over video streams can be modeled as

(ARDigitalObject, `object.type`, {"'gaming.object'"}), where `ARDigitalObject` is a custom-made type intended to model space-sensitive functionality. In addition, a protected space can be defined by a customized attribute of the form (GPSPolygon, `campus`, {"X,Y,W,Z"}) where `GPSPolygon` defines a spatial structure in the form polygon composed of GPS coordinates, which are then labeled as X, Y, W, and Z for illustrative purposes. Attributes may be in turn originated from different sources, e.g., external organizations/ institutions, the S-Apps themselves, or even the supporting devices such as smartphones. As an example, an attribute (String, `app.name`, {"MyApp"}) can be provided by the corresponding S-App itself, and collected at runtime when authorization is requested within a given protected space.

Policies. Using attributes, authorization policies can be constructed for restricting functionality in space-sensitive S-Apps. In our approach, we model access rights (permissions) by combining attributes depicting space-sensitive functionality along with a set of operations that can be performed over them. As an example, a simple <*allow*> operation may be used to model an access right effectively allowing some customized functionality such as the aforementioned (ARDigitalObject, `object.type`, {"'gaming.object'"}). In addition, the protected spaces can be specified using attributes, e.g., the (GPSPolygon, `campus`, {"X,Y,W,Z"}) attribute discussed before. Moreover, policies define the attributes that are required for access rights to be granted or denied. Returning to our running example, a policy may combine the aforementioned attributes along with an attribute depicting affiliation to a given university, e.g., (Membership, `associate.type`, {"Student"}). This way, such a policy will allow for digital objects to be displayed within the protected space comprising a university campus only if the following are met: the requested space-sensitive functionality matches the *object.type* attribute described above, the protected space the end-user is located at a given moment of time can be identified by the *campus* attribute, and the end-user running the S-App happens to be an affiliated student. As shown in Fig. 2, such attributes are to be collected at runtime and forwarded to an implementation framework, along with an authorization request, when the end-user approaches a protected space with an S-App activated on a supporting device.

Attribute Catalogs and Transformations. *SSAC*, as just described, requires a consensus on attribute names, data types, and semantics, so consistency can be guaranteed within the context of a given implementation, e.g., all involved participants should have a clear understanding of the attributes being used, their semantical context, as well as their originating sources, such that the authorization process can be successfully conducted. In such a context, some implementations may require for attributes to be *asserted* by their originating *sources*, e.g., a university digitally signing an attribute depicting student membership, in such a way that both the origin and the integrity of such attributes can be better assessed.

Moreover, our proposed policy model differs from other appraches considering attributes for authorization decisions such as XACML. One of the major differences resides in the lack of support for specifying attribute-based constraints, e.g., requiring the value of a given attribute to be within a certain range. We have based such a design decision in the pursuit of a model that can be easily understood by all participant actors, e.g., policy makers and end-users, etc. However, there may be cases in which such constraints may be required. As an example, consider a policy that requires end-users to be 21 or older of age. Also, assume an attribute named *enduser.age*, which ranges over integer values starting from 0 all the way up to 120 [1], is also available. This way, policy makers leveraging our approach may need to enlist all attribute 3-tuples ranging from 21 to 120 when crafting the aforementioned policy. On the other hand, an alternative approach considering attribute-based constraints may only require an expression of the form "*enduser.age* >= 21", thus significantly improving the convenience for policy makers without sacrificing expressiveness.

In order to overcome these limitations, we propose establishing a dedicated repository, hereafter referred as an *attribute catalog*, which includes standardized definitions for the attributes they provide, e.g., names, data types, as well as a natural language description. In addition, such an *attribute catalog* may also introduce *attribute transformations* that take some attributes as an input and produce some other attributes as a result. As an example, a transformation may be provided such the *enduser.age* attribute can be turned into a (Boolean, `enduser.ismajor`, {"true"}) attribute if and only if the value of the *enduser.age* attribute is equal or greater to 21. These attribute transformations can be in turn implemented in practice as independently-developed modules, e.g., web services, that can be dynamically selected and instantiated at will. Also, they may be provided by either attribute sources, policy makers, or any other trusted participant in the context of an implementation of our approach.

Using attribute catalogs, not only a clear definition of attributes is available, but also enhanced convenience and flexibility is provided. During policy specification time, makers can select the attributes that better describe their authorization needs. In addition, makers may also select a set of attribute transformations that may ultimately produce the attributes listed in their policies. During policy evaluation time, transformations may be automatically instantiated, e.g., invoking a web service, based on the attributes included in the evaluation request. If successful, any produced attributes are returned and used to evaluate the policy as previously described before in this paper.

Attribute Wallets and Bags. Following the policy evaluation procedure depicted in Fig. 2, policy-governed S-Apps, on behalf of end-users, are to collect all relevant attributes and forward them for policy evaluation. In order to facilitate such a process, we propose the introduction of dedicated software modules known as *attribute wallets*, which may be implemented as a submodule of our policy-governed S-Apps or as an independent application running on the end-user's device, e.g., a smartphone. Using such wallets, end-users may be able to store attributes issued on their behalf by sources. Moreover, wallets may also allow for end-users to specify their preferences with respect to a subset of the attributes included within their wallets, which will be ultimately forwarded along with an authorization request. Such subsets, hereafter referred as *attribute bags*, provide end-users with a simple-yet-effective privacy management scheme that may restrict the amount of personal info that gets ultimately shared with policy evaluation engines at a given time.

[1] Or any other reasonable upper bound on human age as considered by the reader.

As an example, the aforementioned attribute depicting (Membership, `associate.type`, {"Student"}), may be issued by a university on behalf of a given end-user, which then stores it inside an attribute wallet. Later on, upon approaching the university's protected space, the end-user may configure an attribute bag to release such an attribute for policy evaluation. This way, previously-defined bags may allow for attributes to be used when evaluating authorization policies, without requiring an interaction with an end-user every single time, assuming attributes remain *valid* over a certain period of time. In addition, in case authorization is denied for a given protected space, end-users may interactively select some other attributes from their wallets, assuming those attributes were not included in previous attempts.

4.2 Collaborative Responsibilities

In order to fully support our policy-governed S-Apps as just described, we envision a collaborative scheme in which different tasks are distributed among participant *actors* in the context of S-Apps, namely, policy makers, application developers, device manufacturers and end-users. Such tasks can be either performed at *runtime*, e.g., when authorization policy is to be evaluated, or *offline*, e.g., when a given S-App is constructed and released to the market. Such tasks are enlisted next.

Policy Makers. First, policy makers should be allowed to specify authorization policies by selecting the attributes identifying the protected spaces under their control, as well as the attributes identifying the S-Apps, developing sources, e.g., companies or organizations, devices, end-users, as well the space-sensitive functionality that is to be authorized or denied. For such a purpose, policies are to follow the authorization model that is described in Sec. 4.3.

S-App Developers. Second, application developers are to provide *S-App-related* attributes such as name, source and version for the S-Apps they produce. In addition, they should also provide a description of the category (or categories) intended for their S-Apps, e.g., games, educational, navigation, etc. Also, at runtime, when the end-user approaches a protected space, S-Apps should also issue authorization requests on behalf of the current end-user, implementing a protocol that includes support for collecting the attributes that will be forwarded as a part of the request, preparing and issuing the request itself, and parsing any received authorization results. Moreover, application developers should make sure their S-Apps enforce authorization decisions, e.g., they only provide space-sensitive functionality as authorized by the evaluated policy.

End-Users. Third, end-users should be allowed to store *end-user-related* attributes such as credentials, memberships, etc., using the attribute wallets introduced in Sec. 4.2. This way, these end-user-related attributes can be eventually forwarded along with an authorization request as described before. Also, end-users should be allowed to create *attribute bags*, depicting the ones that may be ultimately forwarded for authorization as just described. Following the approach for attribute wallets described before, such a selection may be done in an *offline* mode, e.g., before a given S-App is used, thus preventing a detriment of the space-sensitive experience as a result.

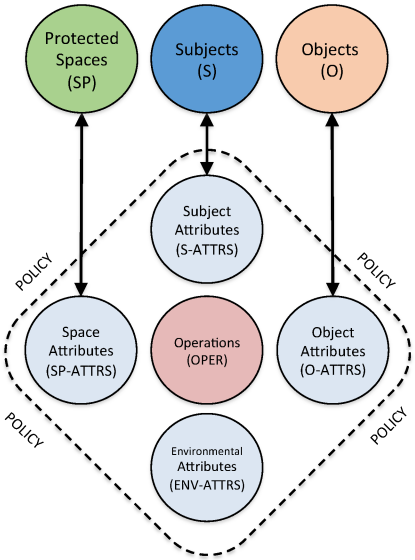


Figure 3: A theoretical authorization model for *policy-governed* S-Apps: the sets of Protected Spaces (SP), Subjects (S) and Objects (O) are mapped to the sets of Attributes: Space (SP-ATTRS). Subject (S-ATTRS) and Object (O-ATTRS), respectively, along with the set of Environmental Attributes (ENV-ATTRS) and the set of Operations (OPER).

Device Manufacturers. Finally, device manufacturers are to provide *device-related* attributes, e.g., brand name, model description, OS name, version, etc., in such a way that those can be later leveraged by policy makers and S-Apps when specifying policies and issuing authorization requests respectively. In the case of policy makers, proper descriptions on the attributes being offered for a given device should be provided by manufacturers, e.g., using the attribute catalog introduced in Sec. 4.1, such that semantically-consistent policies can be produced as a result. For S-Apps, device-related attributes should be available at runtime, so they can be properly located and included as a part of an authorization request.

4.3 Theoretical Model

In order to better describe the ideas introduced in this paper, we now present a theoretical model relating the elements considered in *SSAC* can be combined together to define authorization policies, e.g., how attributes may be used to define a given policy, and how such a policy may be evaluated so access rights can be granted or denied as a result. For illustrative purposes, Fig. 3 presents a graphical depiction of the theoretical model we discuss next.

Composing Elements. Inspired in classical authorization theory, we start by describing the different sets of elements composing our approach. First, we define the set of *protected spaces*, which, as discussed previously in this paper, are physical spaces in which space-sensitive functionality may be displayed and/or restricted, e.g., parks, hospitals, churches, etc. Next, we also introduce the sets of *subjects*, comprising end-users as well as S-Apps and devices, and the set of *objects*, which comprise space-sensitive functionality elements should as visual digital items, video streams, audio files, etc. Finally, we also elucidate the set of *operations*, denoted as

OPER, which either allow or deny the display of the elements in the set of objects as described before.

Attribute Sets. Following the definition of attributes introduced in Sec. 4.1, we now organize them into different sets with respect to the composing elements described in the previous paragraph: first, we consider the set of *space attributes*, denoted as SP-ATTRS, which contains attributes related to the set of protected spaces, and, as hinted before, may be defined as specialized attributes that identify those spaces in the context of a given implementation, e.g., GPS coordinates, location markers, etc., Next, we consider the set of *subject attributes*, denoted as S-ATTRS, which encloses the sets of *device-related*, *end-user-related* and *S-App-related* attributes as discussed in Sec. 4.2. Also, we define the set of *object attributes*, denoted as O-ATTRS, which includes attributes describing space-sensitive functionality, e.g., identifiers, types, categories, etc. and may be ultimately used for selecting space-sensitive functionality to which operations may apply to, as described in Sec. 4.1. Finally, we consider the set of *environmental attributes*, denoted as ENV-ATTRS, which may contain all other attributes not included in the sets above, and may include valuable attribute representations for concepts such as time, counters, etc.

Attribute Equality. Leveraging the definition of attributes we first introduced in Sec. 4.1, we define an attribute a to be equal to another attribute $a\prime$ if the following conditions are met: first, the *data type* and the *name* components of both a and $a\prime$ are the same, and, second, the set of values of a is a subset of the set of values defined for $a\prime$, that is, *values*(a) ⊆ *values*($a\prime$).

Attribute Catalogs. An attribute catalog C = (S-ATTRS, SP-ATTRS, A-ATTRS, O-ATTRS, T) contains the sets of attributes, as well as the set of attribute transformations T (defined below), that are available within a given implementation.

Attribute Wallets and Bags. An attribute wallet $W = (S_W, SP_W, A_W, O_W)$, is a 4-tuple listing subsets of the attributes contained within a given catalog C, such that $S_W \subseteq$ S-ATTRS, $SP_W \subseteq$ SP-ATTRS, $A_W \subseteq$ A-ATTRS, and $O_W \subseteq$ O-ATTRS, An attribute bag $B = S_B \cup SP_B \cup A_B \cup O_B$ is a subset of the attributes contained within a given attribute wallet W, such that $S_B \subseteq S_W$, $SP_B \subseteq SP_W$, $A_B \subseteq A_W$, $O_B \subseteq O_W$.

Attribute Transformations. An attribute transformation is modeled as a mathematical function of the form $t : A \rightarrow A$, where A is a powerset of S-ATTRS ∪ SP-ATTRS ∪ O-ATTRS ∪ ENV-ATTRS. An attribute transformation t takes as an input a non-empty set of attributes, labeled as *input(t)*, and returns a possibly empty set of output attributes, labeled as *output(t)*, as a result.

Attribute Chains. Given a set of attribute transformations $T_C \subseteq T$ and an attribute bag B, an attribute transformation chain, labeled as *chain*(T_C, B), is a partial ordering in T_C such that for $0 \leq i < |T_C|$ and $t_i, t_{i+1} \in$ *chain*(T_C, B), *input*(t_{i+1}) ⊆ B_i, where B_i = *output*(t_i) ∪ B_{i-1} and $B_0 = B$. For a given *chain*(T_C, B), *input*(*chain*(T_C, B)) = *input*(t_0) ∪ B and *output*(*chain*(T_C, B)) = output(t_n) for n = ($|T_C|$ - 1).

Access Requests and Effective Attributes. An access request $R = (B, OP_R)$ is a 2-tuple containing an attribute bag B along with an operation $OP_R \subseteq OPER$. In addition, the set of effective attributes for a given access request $R = (B, OP_R)$ and a set of attribute transformations T_C is given by *effective-attrs*(R, T_C) = B ∪ *output*(*chain*(T_C, B)).

Authorization Policies. A policy *POL* is a 6-tuple listing subsets of attributes contained within an attribute catalog C, such that $POL = (S_P, SP_P, A_P, O_P, OP_P, T_P)$, where $S_P \subseteq$ S-ATTRS, $SP_P \subseteq$ SP-ATTRS, $A_P \subseteq$ A-ATTRS, $O_P \subseteq$ O-ATTRS and $T_P \subseteq$ T. Also, it contains a subset of operations $OP_P \subseteq$ OPER.

Policy Evaluation Strategy. Finally, given a protected space S, a policy *POL* restricting space-sensitive functionality on S, an attribute catalog C = (S-ATTRS, SP-ATTRS, A-ATTRS, O-ATTRS, T), and an access request R on S, *Grant-Access*(S, *POL*, R) is a boolean function that evaluates to `True` iff $OP_P = OP_R$ and $S_P \cup SP_P \cup A_P \cup O_P \subseteq$ *effective-attrs*(R,T_C) for some $T_C \subseteq T$. Otherwise, *Grant-Access*(S, *POL*, R) returns `False` as a result.

5 EXPERIMENTAL EVALUATION

In this section, we start by describing a supporting authorization framework as well as a proof-of-concept S-App, which we later leverage for providing a case study tailored to evaluate the runtime performance of *SSAC*. Later, we also describe a user study that included the exposure of several participants to our approach, ultimately resulting in evidence that supports the suitability of our proposed approach to be successfully implemented in practice.

5.1 Proof-of-Concept Implementation

Our supporting authorization framework includes a dedicated API for handling both policy administration and runtime policy evaluation requests, allowing for the former to be used in an *offline* mode by policy makers to properly add, change and remove policies, whereas the latter may be used by S-Apps to issue authorization requests at runtime when entering a given protected space. Moreover, our framework also includes a set of remote modules, which implement policy storage and evaluation tasks, thus effectively reliving S-Apps from having to implement those from scratch.

Protected Spaces. We model protected spaces by leveraging the Google Maps API [11], e.g., a given protected space is represented as a set of GPS coordinates, a.k.a., a geo-spatial *polygon*. Each protected space is also identified by a location *marker*, which is then placed within the boundaries defined by its corresponding polygon. Our framework leverages the Google Maps API to allow for policy makers to interactively select their protected spaces using its map-like *graphical user interface* (GUI) capabilities to define both polygons and location markers.

Authorization Requests. The authorization S-App-side API depicted by our framework includes a set of Java method calls implementing authorization requests, a.k.a., *authorization checks*, which, as mentioned in Sec. 4.2, are to be placed by developers within the source code of their S-Apps. As an example, an authorization check of the form

```
Auth.request(new MARDigitalObject("pokestop"), bag);
```

may be used to request or authorization to display digital objects known as *pokestops*, following the running example shown in Fig. 1.

Proof-of-Concept S-App. We have also implemented a *proof-of-concept* gaming S-App, called `SpaceProtector`, whose snapshot is shown in Fig. 4, in which end-users are encouraged to explore their physical surroundings while capturing digital objects that are displayed on top of a video stream, featuring an scenario similar

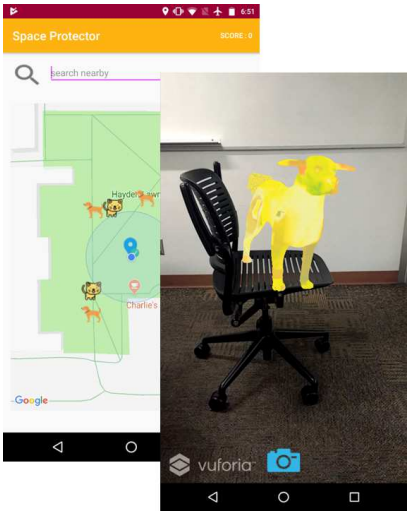


Figure 4: `SpaceProtector`: A *proof-of-concept* S-App for *SSAC*.

to the one introduced in Figs. 1 and 2. `SpaceProtector` also leverages Google Maps for geo-spatial operations, modeling protected spaces as sets of GPS coordinates, and provides an attribute catalog as well as an attribute wallet to assist end-users in writing authorization policies and defining custom attribute bag configurations. Finally, `SpaceProtector` handles authorization requests transparently to end-users by means of a *back-end* service implementing *SSAC*, following the approach discussed in Sec. 4.

Policy Evaluation. In addition, our framework provides a *policy decision point* (PDP) module, which implements the policy evaluation strategy discussed in Sec. 4.3. For such a purpose, S-Apps continuously communicate their physical location such that the current protected space, if any, can be identified, leading to the subsequent evaluation of all relevant policies once authorization is requested. Such an strategy may become a performance bottleneck if the physical location of the end-user changes rapidly with respect to the functionality depicted by the S-App, and several authorization requests may be issued as a consequence. In order to prevent this situation, our framework implements an *eager* policy evaluation strategy in which the policies related to nearby protected spaces are fetched and evaluated beforehand, thus allowing for S-Apps to store the policy evaluation results of nearby protected spaces and use them when needed. We implement this *eager* strategy by leveraging our so-called *policy evaluation windows*: a set of GPS coordinates depicting a geo-spatial polygon surrounding the current physical location of the end-user. This way, at a given moment of time, the policies related to all the protected spaces overlapping a given window are retrieved and evaluated in advance. Once the current location changes beyond a certain pre-defined distance, the window is recalculated and the policies of previously-unseen spaces are evaluated as well.

5.2 Performance Case Study

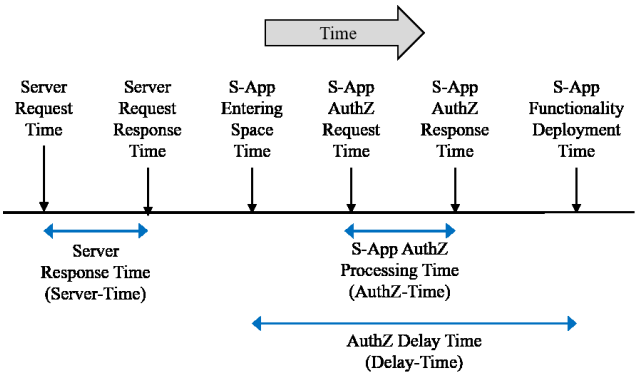
Description. Using `SpaceProtector`, as well as our supporting authorization framework implementing *SSAC*, we conducted a controlled case study simulating an end-user playing with such an S-App over a university campus, which was in turn enhanced with a predefined set of protected spaces representing important buildings and a rest area, as it is graphically shown in Fig. 5 (a). In addition, each protected space was augmented with an authorization policy closely resembling realistic scenarios tailored for each space. As an example, academic buildings obtained policies restricting, either partially or totally, the use of gaming S-Apps, whereas their use was fully allowed at rest areas.

Timestamps. In order to collect runtime performance information during our study, we instrumented the source code of both our `SpaceProtector` S-App to obtain the following timestamps: the *entering space timestamp* (Entering-Stamp, for short), which, as the name implies, is obtained once an end-user, along with the corresponding device, enters a given protected space by the first time having our S-App activated. In addition, we both obtained the *authorization request timestamp* (S-App Request-Stamp), which is obtained once the S-App requests our supporting framework for authorization to display space-sensitive functionality, as well as the *authorization response timestamp* (S-App Response-Stamp), once a response containing an authorization decision is received. Then, we also took the *functionality deployment timestamp* (Deployment-Stamp), which represents the moment of time in which the S-App renders or denies space-sensitive functionality based on the authorization decision(s) just received. Conversely, we also instrumented our authorization framework to obtain the following timestamps: the *server request timestamp* (Server-Request-Stamp) and the *server response time* (Server-Response-Stamp), which, as their names imply, capture the moment when an authorization request is received and subsequently served by sending a response back on a given S-App.

Metrics. Using the timestamps just discussed, we introduced the following performance metrics: initially, we calculate the *server response time* (Server-Time) as the difference between the Server-Response-Stamp and the Server-Request-Stamp. In addition, we calculated the *authorization processing time* (AuthZ-Time) as the difference between the Response-Stamp and the Request-Stamp defined for S-Apps. In a similar approach, we also calculated the *authorization delay time* (Delay-Time), which models the overall delay taken by an S-App to render space-sensitive functionality once entrance to a new protected space has been detected. Such a metric was modeled by subtracting the Entering-Stamp from the Deployment-Stamp defined for S-Apps. The relationship between the aforementioned performance timestamps and metrics is graphically described in Fig. 5 (b).

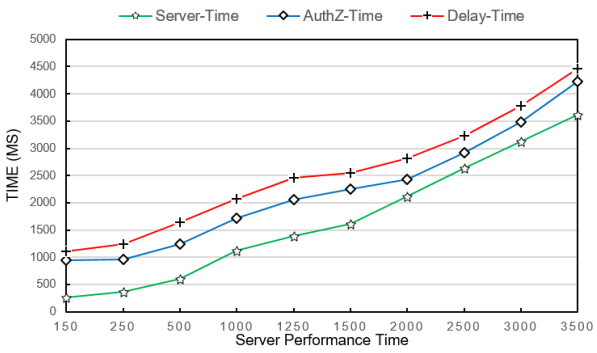
Modes. For our case study, we also defined three different usage modes for our `SpaceProtector` S-App, in an effort to better assess the capabilities depicted by *SSAC* as well as to provide a fair platform for comparison when it comes to runtime performance. First, we defined a *no-authorization mode* (No-AuthZ Mode), which, as suggested by its name, implied the use of `SpaceProtector` without any authorization-based functionality activated, that is, space-sensitive functionality is automatically rendered as soon as a new protected space is detected, thus ultimately simulating the standard runtime behavior exhibited by many S-Apps currently available in the market. Leveraging the discussion introduced in the previous paragraph, our proposed No-AuthZ Mode depicted the Server-Time as well as the AuthZ-Time all set to zero, as such functionality was not provided as just mentioned. Second, we also

(a) An experimental testbed setting for SpaceProtector.

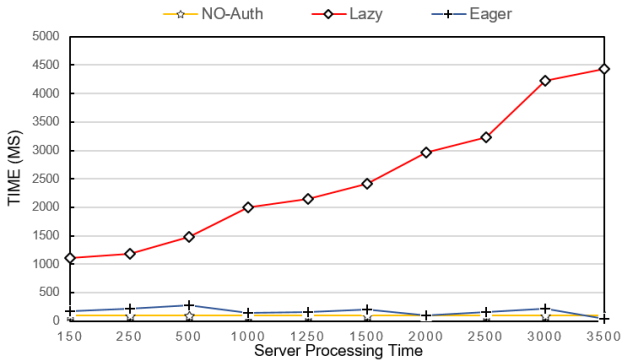


(b) Timestamps and metrics for evaluating S-Apps.

Figure 5: Experimental Testbed for Evaluating the Runtime Performance of SpaceProtector.



(a) Metrics for the Lazy Usage Mode.



(b) Comparing Different Usage Modes.

Figure 6: Experimental Results Depicting Runtime Performance of SpaceProtector.

defined an *authorization lazy mode* (Lazy Mode), which includes SpaceProtector requesting our supporting framework for authorization before rendering any space-sensitive functionality every single time entrance to a new protected space is detected. Finally, we also implemented an *eager authorization mode* (Eager Mode) that besides leveraging the features of our supporting authorization framework, also features the *policy evaluation windows* previously described in Sec. 5.1 to dynamically retrieve authorization decisions for nearby protected spaces and store them into a dedicated cache structure implemented by SpaceProtector, thus avoiding frequent lengthy *round-trips* to our supporting authorization framework as depicted in the Lazy Mode just discussed. During the Eager Mode, all metrics, as discussed in previous paragraphs, were calculated.

Procedures. As described before in this Section, our case study involved an End-User traversing a series of protected spaces determined by a university campus landscape. In such a process, the simulated end-user would advance through a walking aisle as depicted in Fig. 5 (a), entering protected spaces once at a time. Upon entering a new space, authorization for delivering space-sensitive functionality would be requested (for the Lazy and Eager modes), whereas no authorization would be performed for the No-AuthZ Mode. When appropriate, the performance metrics also discussed before will be collected. In such a context, the following parameters were set beforehand: For the Lazy and Eager Modes, as well as for each protected space, a single authorization policy, encoded in the format described in Sec. 4 was defined, requesting for 4 attributes as well as 1 attribute transformation before space-sensitive functionality can be ultimately delivered to End-Users. In addition, for the Eager Mode, the size of the *policy evaluation window* was set to 20 meters, refreshing the authorization results stored in the internal cache implemented by SpaceProtector once the end-user covers such a distance when walking. In addition, for the Lazy and Eager Modes, we defined a variable parameter called *server-side processing time* (Server Processing Time), which was intended to provide a comprehensive modeling of the authorization process performed by our supporting framework in the presence of large number of stored policies as well as a large number of concurrent authorization requests being made by S-Apps. With that in mind, such a Server Processing Time subsumes tasks related to the location and retrieval of authorization policies, the evaluation of such policies and the communication of the authorization decisions, as well as the overall overhead caused by many different concurrent requests, as mentioned before. For our case study, we

hosted our supporting authorization framework in a Dell Laptop running Windows 10, with 16 GB of RAM and 1128 GB of HD storage. In addition, our `SpaceProtector` S-App was hosted in a Moto G4 mobile device running Android 7.0, with 2 GB of RAM and 16 GB of HD storage. Finally, we performed our case study during the course of three days, performing 10 instances, e.g., a walk-through traversing all the protected spaces from left to right as depicted in Fig. 5 (a) each time. We obtained all the timestamps and metrics related to the aforementioned usage modes, and calculated the average for all of them.

Results. Our results can be described as follows: Fig. 6 (a) shows the results for the Lazy Usage Mode. As expected, the metrics Server-Time, AuthZ-Time and Delay-Time show an increment that is proportional to the increase in the server performance time, thus affecting the overall performance of our `SpaceProtector` S-App, as end-users may be able to experience a noticeable delay in the rendering of space-sensitive functionality. Finally, Fig. 6 (b) shows the Delay-Time metric for the three usage modes being evaluated. As shown, the Lazy Mode incurs in the most significant increment, which is proportional to the time taken by the authorization framework to process requests at runtime every time a new protected space is entered. In addition, the No-Auth and the Eager Modes incur in a comparable performance detriment, as denoted by the similar results obtained for the Delay-Time metric despite increments in the server processing time, providing evidence supporting the adoption of the Eager Mode by future S-Apps.

5.3 User Study

As mentioned in Sec. 3, *SSAC* may be ultimately used by individuals (either space owners or end-users) who may or may not have received previous training in authorization. With that in mind, we designed a user study to assess the policy understanding and writing of participants that may eventually become either policy makers (e.g., space owners) and end-users (e.g., gamers). Such a study included the following research questions (RQ):

RQ1 Can participants understand drafted authorization policies using *SSAC*?.

RQ2 Can participants write policies using *SSAC*?.

Our study was formally approved by an institutional review board (IRB) office which oversees research integrity and ethics. In the rest of this section, we present the experimental design as well as the results of our study, which provide evidence of the suitability of our approach for supporting the research questions just described.

5.3.1 Participants. For our study, we recruited 30 participants by placing an invitation script in different locations throughout a university campus. From this sample, we were able to recruit 10 participants having some background in computer science (CS) technologies, e.g., they were enrolled in a CS-related major, and 20 participants who self-reported having no formal background in CS-related technologies or educational programs. Having no formal background in CS-related studies was important to the study in order to assess the potential for non-CS end-users.

5.3.2 Procedure. Our study was conducted in 5 sessions involving 6 participants each. Sessions were held in a seminar room at a university campus in the course of a week, one session per day. Each session took about 70 minutes. Each participant received an Amazon voucher for $20.00 USD for their participation. Each session of the empirical evaluation consisted of 4 steps: introduction and demonstration (25 minutes), policy understanding (15 minutes), policy writing (15 minutes) and survey (15 minutes). Each step is summarized in the following.

Step 1: Introduction and Demonstration. In the introduction, participants were presented an overview of the approach, similar to Sec. 4. A short visual demo featuring the implementation framework as well as the *proof-of-concept* `SpaceProtector` S-App described earlier in this section was shown. Then, participants were informed about the next steps: (2) policy understanding, (3) policy writing and (4) survey.

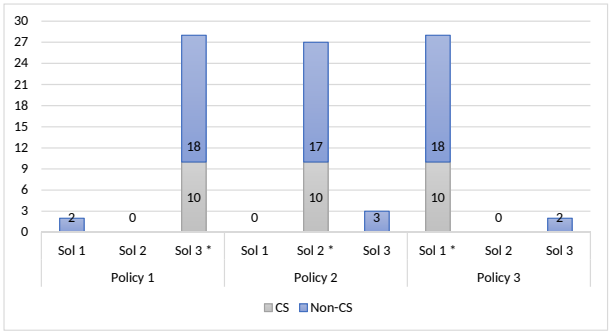
Step 2: Policy Understanding. The second step aimed to address RQ1. We provided three space-sensitive authorization policies encoded. For each policy, we provided three possible solutions, and asked participants to select the one that most accurately resembles the meaning of the authorization policy. One of the solutions was specially crafted to be an exact match, thus depicting the right answer to the question. The other two solutions were intentionally modified to be either more *permissive*, or more *strict*, e.g., by deliberately allowing/denying S-Apps to display more/less functionality within the protected space. An example policy is shown in Table 1.

Table 1: Policy Understanding. Option (1) is the correct one.

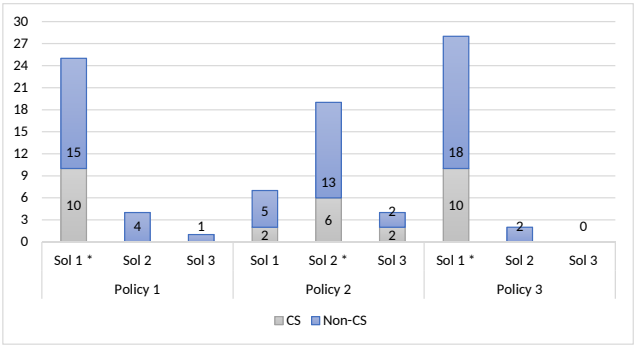
Policy 3	
<*allow*> (PSpace, `space.location`, {"Museum"}) (String, `app.category`, {"History"}) (*space.GPSCoordenates*) → (*space.location*) (*app.name*) → (*app.category*)	
Responses	
1	*The local museum wants to allow only apps having category "History" within its facilities.*
2	*The local museum wants to deny all apps depicting any kind of content within its facilities.*
3	*The local museum wants to allow all apps within its facilities.*

Table 2: Policy Writing. Option (2) is the correct one.

Policy 2	
"An alcoholic beverage producer wants to restrict its S-App AlcoApp only to users who are equal or above 21 years old and physically located in the US".	
Responses	
	<*deny*> (`space.country`, {"USA"}) (`user.isMajor`, {"True"}) (`app.name`, {"AlcoApp"}) (*user.age*) → (*user.isMajor*)
	<*allow*> (`space.country`, {"USA"}) (`user.isMajor`, {"True"}) (`app.name`, {"AlcoApp"}) (*user.age*) → (*user.isMajor*)
	<*allow*> (`space.country`, {"USA"}) (`user.isMajor`, {"True"}) (`app.manufacturer`, {"Western Drinks"}) (*user.age*) → (*user.isMajor*) (*app.name*) → (*app.manufacturer*)



(a) Step 2: Policy Understanding.



(b) Step 3: Policy Writing.

Figure 7: User Study Results. Correct Solutions are depicted with an Asterisk (*).

Step 3: Policy Writing. This step aimed to address RQ2 on policy writing. We provided three descriptions of space-sensitive authorization policies, and, for each of them, we provided three candidate encodings. This time, participants were required to select the encoding that would better match the description of each policy. Similar to the previous step, we included a *correct* encoding accurately matching the policy semantics, and included two other encodings, a *permissive* and a more *strict* one. Participants were provided with a list of operations and attributes to choose from, thus resembling an *attribute catalog* as it was described in Sec. 4 as well. An example policy is shown in Table 2.

Step 4: Survey. Finally, participants were asked to fill out a survey containing five questions regarding (1) their familiarity with S-App technologies, (2) their acceptance for implementing authorization techniques in the context of S-Apps, (3) their interest in implementing the proposed approach for authorization in the context of S-Apps, (4) their perception on well they believe they understood our proposed approach and (5) further comments, remarks and improvements to the approach.

5.3.3 Results. We summarize the findings of the study next.

Participants Background and Appreciation. The background and some results of the survey can be described as follows: first, 67% of all participants expressed a moderate to high degree of familiarity with S-App technologies. Hence, more than two thirds have had already experience with S-App technologies. As described in Section 5.3.1, only one third had a CS background (10 out of 30). Second, 87% of all participants agreed there is a need for authorization mechanisms in the context of space-sensitive S-App scenarios. In addition, 93% of all participants expressed an interest in restricting functionality of S-Apps inside a given protected space. Hence, the participants recognized the need for authorization mechanisms in S-App scenarios. Even more, they would be interested to use and apply such an approach. This supports our ideas and proof of concept that there is a need for authorization in S-App scenarios and users would be interested in it. Finally, 83% of all participants expressed a high degree of understanding of our proposed approach, which complement the results obtained for Steps 2 and 3 and provide additional evidence of the suitability of `SpaceProtector` for being deployed in practice.

Based on these highly motivating results, lets take a look into the detailed results of Steps 2 and 3.

RQ1: Can participants understand policies? To answer RQ1, we conducted Step 2 on policy understanding. The results are shown in Fig. 7 (a). It can be seen from the figure that most participants were able to select the correct answer for all three depicted policies. Participants with CS background chose the correct solution all the time, whereas Non-CS participants did select a few incorrect answers. In total, more than 85% of the participants could identify the correct answer for all three policies.

RQ2: Can participants write policies? Similar results were obtained for RQ2 on policy writing as shown in Fig. 7 (b). It can be seen from the figure that the majority of participants were able to select the correct solution. This time, however, a few CS and Non-CS participants failed to select the correct solution for Policy 2, shown in Table 2. This policy was more complex as it required a selection of both attributes and transformations which might have been more challenging for the participants. On the other hand, the formulation of the policy itself *"..wants to restrict its S-App AlcoApp.."* might have led participants to believe either to deny (Solution 1) or allow (Solution 2) this certain policy. Solution 3 refers to *"Western drinks"* which are not mentioned in the formulated policy. However, to fully understand the motivation more research and tests for policy writing are required. Finally, the findings of the user study support the assumptions, expressed earlier in this paper, that end-users can understand and write authorization policies for space-sensitive scenarios, following the challenges that have been devised in previous work [14]. Further research and investigation has to be conducted for formulating policies in natural language and how to map them into specific attributes.

6 DISCUSSION AND FUTURE WORK

Space Ownership. As introduced in Sec. 3 and Sec. 4.1, we aim for space owners to specify policies restricting space-sensitive functionality over their corresponding protected spaces, which then introduces the question of who can be regarded as the *rightful* owner of a given physical space. Whereas such a topic has been deliberately left out of the scope of this paper, we believe space ownership may be established *offline* by an authority entity, who besides being able to assess the identity of the space owner, may

also effectively locate the boundaries of a given protected space, e.g., by providing any attributes that may better identify it, and is also able to resolve *multi-tenancy* issues, e.g., two or more organizations/individuals claiming ownership over the same space. Future work may then focus on implementing a solution for determining space ownership as just described, at the same time it also supports delegation, e.g., property landlords delegating to renters, or organizations delegating to some departments control over a certain protected space.

Attribute Catalogs and Transformations. In addition, future work may also focus on extending our proposed attribute catalogs to include more attribute descriptions as well as transformations. As it is described in Sec. 4.2, we expect other collaborative organizations to make such attributes available for the generation of both attribute catalogs and transformations, thus ultimately resulting in a large variety of options policy makers may choose from. In such regard, Google has recently transitioned into a dedicated security framework called BeyondCorp [24], which leverages pieces of security-related information, similar to our proposed attributes, for making authorization decisions. Based on this, we believe the use of attributes for authorization decisions is in its way to become widespread, as organizations will certainly appreciate the benefits provided by their convenience and flexibility, as we have discussed previously in this paper.

Policy Enforcement. As shown in Sec. 4, *SSAC* allows for developers to take into account potential authorization decisions to regulate functionality within their S-Apps, which requires developers to insert proper API calls within their source code when appropriate. In such a context, *SSAC* introduces a convenient alternative for automatically handling requests for restricting space-sensitive functionality, which are currently mostly handled on a semi-automated, *one-by-one* basis [18]. This way, developers would be relieved from having to provide dedicated authorization infrastructure from scratch, and space owners would be also relieved from having to learn a new paradigm and repeat the process for every S-App that hits the market and becomes relevant. Future work may then focus on providing enhanced support for policy enforcement by inspecting technologies for content filtering at the OS level, allowing for our approach to be efficiently integrated with existing and newly-developed S-Apps, while still maintaining enough flexibility so developers can decide how they will react to authorization decisions at runtime.

7 CONCLUSIONS

In this paper, we have elaborated on the security challenges imposed by emerging S-App technologies, including the need for space owners as well as application developers to specify and enforce restrictions on what space-sensitive functionality is acceptable under certain physical spaces. In order to address such challenges, we have proposed a collaborative authorization framework integrating organizations, application developers, device manufacturers and end-users, providing rich and flexible support to handle different cases and needs by leveraging attribute-based policies. Finally, as depicted by the encouraging results obtained during the studies we have presented as a part of this paper, we believe our approach may be able to accommodate for other emerging space-sensitive technologies that are introduced in the future.

ACKNOWLEDGMENTS

This work was partially supported by grants from the National Science Foundation (NSF-IIS-1527268 and NSF-ACI-1642031).

REFERENCES

[1] C. A. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. 2006. Supporting Location-based Conditions in Access Control Policies. In *Proc. of the Sym. on Info., Computer and Communications Security (ASIACCS '06)*. ACM, New York, NY, USA, 212–222.

[2] C. A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati. 2009. Access Control in Location-Based Services. In *Priv. in Location-Based Applications*. Number 5599 in Lecture Notes in Computer Science. Springer, 106–126.

[3] Steffen Bartsch. 2011. Authorization Enforcement Usability Case Study. In *Engineering Secure Software and Systems*, Ú. Erlingsson, R. Wieringa, and N. Zannone (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 209–220.

[4] BBC. 2017. Pokemon Go away: Troublesome Sydney Pokestop shut down. http://www.bbc.com/news/technology-36948331. (2017). [Online; accessed June-22-2017].

[5] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, and Misa Ivkovic. 2011. Augmented Reality Technologies, Systems and Applications. *Multimedia Tools Appl.* 51, 1 (Jan. 2011), 341–377.

[6] Daily Sabah Europe. 2017. Germany's Auschwitz-Birkenau Museum says no Pokemon Go. http://www.dailysabah.com/europe/2016/07/14/germanys-auschwitz-birkenau-museum-says-no-pokemon-go. (2017). [Online; accessed June-5-2017].

[7] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. 2007. GEO-RBAC: A Spatially Aware RBAC. *ACM Trans. Inf. Syst. Secur.* 10 (Feb. 2007).

[8] M. Decker. 2008. Requirements for a Location-based Access Control Model. In *Proc. of the 6th Int. Conf. on Advances in Mobile Computing and Multimedia (MoMM '08)*. ACM, New York, NY, USA, 346–349.

[9] Facebook Inc. 2018. Facebook for Developers. https://developers.facebook.com/. (2018). [Online; accessed November-22-2018].

[10] M. Fernández and B. Thuraisingham. 2018. A Category-Based Model for ABAC. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control (ABAC'18)*. ACM, New York, NY, USA, 32–34.

[11] Google Inc. 2017. Google Maps API. https://developers.google.com/maps/. (2017). [Online; accessed July-14-2017].

[12] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. 2014. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication* 800 (2014), 162.

[13] S. Jana, D. Molnar, A. Moshchuk, A. Dunn, B. Livshits, H. J. Wang, and E. Ofek. 2013. Enabling Fine-grained Permissions for Augmented Reality Applications with Recognizers. In *Proc. of the 22Nd USENIX Conf. on Security*. 415–430.

[14] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner. 2018. Towards Security and Privacy for Multi-user Augmented Reality: Foundations with End Users. In *2018 IEEE Symposium on Security and Privacy (SP)*. 392–408.

[15] Lime, Inc. 2018. Lime Dockless Electric Scooter Share. https://www.li.me/electric-scooter. (2018). [Online; accessed November-26-2018].

[16] M. Miettinen, S. Heuser, W. Kronz, A. Sadeghi, and N. Asokan. 2014. ConXsense: Automated Context Classification for Context-aware Access Control. In *Proc. of the ACM Symp. on Info., Comp. and Comm. Sec. (ASIA CCS '14)*. ACM, 293–304.

[17] Niantic, Inc. 2017. Pokemon GO. http://www.pokemongo.com/. (2017). [Online; accessed June-5-2017].

[18] Niantic, Inc. 2018. Request Modification for Pokemon GO. https://goo.gl/uSZ1hP. (2018). [Online; accessed November-28-2018].

[19] OASIS Standard. 2013. eXtensible Access Control Markup Language (XACML) Version 3.0. (2013, January 22). (2013). http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[20] Q. M. Rajpoot, C. Damsgaard Jensen, and R. Krishnan. 2015. Attributes Enhanced Role-Based Access Control Model. In *Int. Conf. on Trust, Privacy and Security in Digital Business (TrustBus)*.

[21] F. Roesner, D. Molnar, A. Moshchuk, T. Kohno, and H. J. Wang. 2014. World-Driven Access Control for Continuous Sensing. In *Proc. of the Conf. on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 1169–1181.

[22] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. 1996. Role-Based Access Control Models. *Computer* 29, 2 (Feb. 1996), 38–47.

[23] Time. 2017. Pokemon Go Players Anger 9/11 Memorial Visitors. http://time.com/4403516/pokemon-go-911-memorial-holocaust-museum/. (2017). [Online; accessed June-5-2017].

[24] R. Ward and B. Beyer. 2014. BeyondCorp: A New Approach to Enterprise Security. *;login:* Vol. 39, No. 6 (2014), 6–11.