THESIS

ACCESS CONTROL FOR IOT ENVIRONMENTS: SPECIFICATION AND ANALYSIS

Submitted by

Jordan T. Peterson

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2021

Master's Committee:

Advisor: Dr. Indrakshi Ray
Co-Advisor: Dr. Vinayak Prabhu

Dr. Joeseph Gersch
Dr. Stephen Hayne

ABSTRACT

ACCESS CONTROL FOR IOT ENVIRONMENTS: SPECIFICATION AND ANALYSIS

Smart homes have devices which are prone to attacks as seen in the 2016 Mirai botnet attacks. Authentication and access control form the first line of defense. Towards this end, we propose an attribute-based access control framework for smart homes that is inspired by the Next Generation Access Control (NGAC) model. Policies in a smart home can be complex. Towards this end, we demonstrate how the formal modeling language Alloy can be used for policy analysis. In this work we formally define an IoT environment, express an example security policy in the context of a smart home, and show the policy analysis using Alloy. This work introduces processes for identifying conflicting and redundant rules with respect to a given policy. This work also demonstrates a practical use case for the processes described. In other words, this work formalizes policy rule definition, home IoT environment definition, and rule analysis all in the context of NGAC and Alloy.

ACKNOWLEDGEMENTS

This work is supported in part by funds from NIST under Contract Number 60NANB18D20, NSF Awards CNS 1650573, CNS 1822118 and funding from CableLabs, Furuno Electric Company, SecureNok, Statnett, Cyber Risk Research, and AFRL.Research findings and opinions expressed are solely those of the authors and in no way reflect the opinions of the NSF or any other federal agencies.

First I want to show my gratitude to Dr. Indrakshi Ray for mentoring and advising me throughout my time in higher education. She has supported me and challenged me to grow academically since I first met her. Without her encouragement I would not have expected to progress into the graduate school. Furthermore the experience I have gained through her teaching and IoT Security Laboratory have enabled me to pursue a career in cyber security.

I also thank Dr. Vinayak Prabhu. His involvement and dedication to this work have been paramount. I appreciate the time he spent with me on the many meetings to iron out each important detail. I am also grateful for wealth of knowledge he has provided as a resource to this project.

I would like to thank Sophia Ressler for her support on this work. The direction and scope of this work was greatly impacted as a result of our many discussions. Her collaboration with Margrave and Alloy also shows in this work.

Thank you Tim Nelson for taking the time to learn about this work and steer me in the direction of Alloy. He was also very helpful when I first began this project by volunteering his time to help me better understand the tool Margrave. Although we ultimately didn't use this tool in the final product, his guidance was integral to our utilization of Alloy.

Lastly, I want to acknowledge Kendra Peterson. Her willpower and dedication towards life has been inspiration for my every step of this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Access control management systems are used extensively in enterprise networks. However, access control is a relatively new topic for the Internet-of-Things (IoT) environments. Consider a smart home environment which uses a large number of proven insecure smart devices. The use of insecure devices connected to the internet can be compromised by malicious actors and cause damage to unsuspecting public organizations. This was demonstrated by the Mirai botnet attacks of late 2016. In [1] Antonakakis et al. describe the four separate distributed denial of service (DDoS) attacks on OVH, Krebs on Security, Dyn, and Liberia. However, these are just the high profile targets, a total of 15,194 different attacks were attributed to Mirai between September 27, 2016 and February 28, 2017. The propagation of this bot constructing malware took advantage of default credentials to gain access to a steady state of 200,000 - 300,000 default or misconfigured IoT devices. Utilizing Censys [2] Anatonakakis et al. were able to identify Mirai infected devices and found a peak of 600,000 compromised devices. The sheer volume of devices at Mirai's disposal lead to record breaking attacks on the four high profile targets mentioned at the beginning of this section. Krebs on Security experienced a 623 Gbps DDoS attack. Dyn reported DDoS attacks lasting one, five, and ten hours long involving 70,000 Mirai bots. Liberia experience a deterioration of internet accessibility which is speculated to be from the 616 attacks performed by Mirai on Lonestar Cell, a large telecom operator in Liberia. This case is interesting as it shows that these mass IoT compromises impact more than just the United States as the internet is a global construction. Though the potential of Mirai was not met with a coordinated attack involving all of its resources, the havoc introduced was possible at a period of relative infancy for the IoT market. Gartner reported that in 2016 there was an estimate of about 6 billion IoT devices were installed worldwide [3]. Current estimations by Gartner for the number of IoT devices installed by 2020 is up to 20 billion [4]. With such rapid growth in the IoT industry the potential for devastating coordinated attacks needs to be addressed. Our work aims to mitigate or reduce the potential for mass

disruption through providing a defined process of verifying attribute-based access control (ABAC) policies through the popular specification language Alloy [5].

Due to the nature of malware propagation, one way to reduce infection throughout a network is to restrict communication between devices, essentially segregating the network. Our most powerful concept for defining and managing these restrictions is access control (AC). The concept of creating rules to govern the behavior of our networks is essential to managing the health of our networks and broadly the health of the internet. However, AC policies can be large and complex with many rules of varying scope orchestrating how a network of devices behaves. As humans are prone to mistakes when working with large or complex systems it is necessary to verify said policies. In addition to the technical nature of AC management, a large percentage of IoT devices are consumer products being installed into the common household. These Smart-home environments are prone to non-technical administrators who are either more likely to have misconfigured network security, or none at all. This work does not intend to solve this specific issue, but is anticipating greater involvement of AC in home environments. In this work we provide demonstration of a core policy verification process involving an emergent ABAC scheme that we have modified to fit the IoT home environment context. We prepare for an IoT-centric access control scheme and management solution to protect the emergent wave of smart homes. Potential sources of such control are local internet service providers (ISP) or home router integrated software. That being said we do not provide any designs for effective access control management user interfaces that would be required to address the non-technical nature of the average home owner.

As discussed costly damage can be done when IoT devices are compromised and perform malicious actions. On the other hand, damage can also be done if IoT devices are restricted from performing actions they were designed for. This is especially relevant to the medical IoT community. There are a increasing number of home and industrial medical devices that are being installed that are expected to perform without flaw in order to ensure the health of the users. If these devices fail to provide their designed functionality people could be hurt and potentially fatal incidents may occur. One such cause of functional failure is logical mistakes within an access control policy.

Imagine a scenario where a patient is dependant on the function of a medical IoT device or system of devices. An administrator is configuring an access control policy to help secure device communication within the network. The administrator explicitly states that the medical device should have sufficient access within the network to perform correctly. However, unknowingly there is a conflict between the existing policy and this new statement. There is no conflict identification in place when creating the access control policy and as a result the conflict is overlooked by the administrator with the assumption that the conflict does not exist. The new policy is then enacted and the medical device loses sufficient access and in turn the required functionality to help the patient. This could be prevented if a system was in place for verifying the policy as it was being created. This is one situation that our work directly addresses.

Our work utilizes a modification of the Next Generation Access Control (NGAC) [6] framework as our ABAC base. The leading authorization system that leverages NGAC is Policy Machine [7], both designed by NIST. Policy Machine is able to derive capabilities of entities, in other words what the entity can do. It provides administrative operations, which are functionality for altering the policy. It also provides enforcement functions, which are boolean equations used to make decisions about access. However, Policy Machine is unable to reason about the policy. Our solution leverages a specification language named Alloy [5] in order to identify access control policy redundancies and conflicts. Unfortunately, NGAC has no supporting tools for policy analysis at the time of this work. This is one of our motivations for utilizing Alloy. This however poses an issue regarding representing the NGAC policy in Alloy due to there being no translation tools either. In order to achieve this step we designed a NGAC-to-Alloy translation algorithm. This allows us to take any NGAC policy definition and create a policy representation within Alloy. Once the policy is defined in the Alloy language we can perform conflict/redundancy identification on proposed rules with respect to said policy. This analysis allows us to detect conflicts between proposed rules and an existing policy. Alloy also enables us to detect redundancy between proposed rules and an existing policy.

This work provides novel contributions to the IoT, Access Control, and Formal Modeling research communities. We provide a verification process from start to finish beginning with environment and policy definition and ending with conflict and redundancy identification. This process utilizes existing tools such as Alloy Analyzer and access control frameworks such as NGAC which we do not claim to have created. We define and demonstrate translation between NGAC policies / IoT Environments to Alloy specification language. We define a methodology for detecting and identifying conflicts and redundancies between newly added rules and an existing policy. Finally, we provide a practical example of this process in action and the security benefits it provides the operators stated in Chapter 9

1.1 Research Organization

This work follows a process defined to organize and expedite the modeling of our example IoT environment and smart home network access control policy. This process is depicted in Figure 1.1 which will be described in brief for this section as well as in detail throughout the work where applicable.

The process illustrated in Figure 1.1 begins with defining the IoT environment. This environment is composed of a multitude of devices which abide by real world limitations and rules. These devices, limitations, and rules create a model of the universe in which access control is being applied. In short, by defining the IoT environment we enumerate, along with other important aspects, the subjects, objects, and access rights that are formed into rules for a policy. This step is further described in Chapter 5.

After the environment has been defined, elements from our now defined working universe can be composed into rules and collected into a rule set. We depict this as the IoT environment *influencing* the rule set definition. From this rule set we can extract access control components such as entity relationships, allowances and denials, and specified constraints. These components are integral to defining the NGAC framework inspired access control policy for this work. This step is further described in Chapter 6.

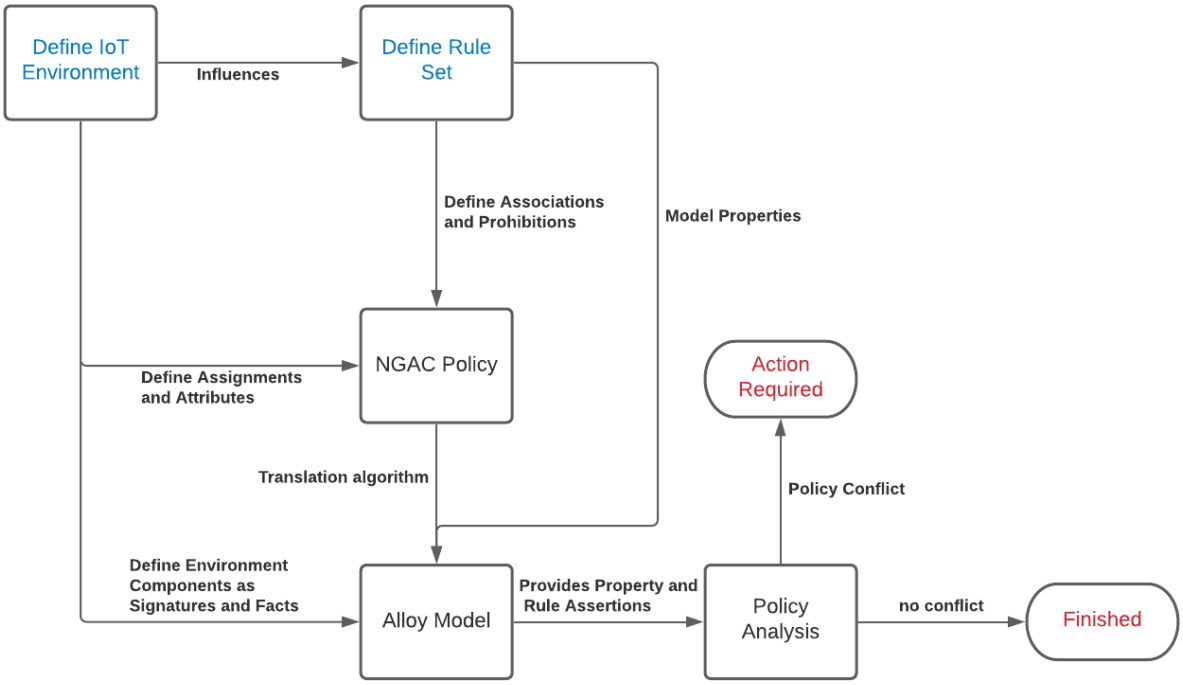

Figure 1.1: NGAC Policy Verification Process

With the rule set and environment defined we have all the information required to construct our NGAC inspired policy. From the environment we extract the assignments and attributes making them the backbone of the policy. From the rule set we extract the necessary components for creating associations and prohibitions. There are interesting nuances regarding IoT environments which resulted in a modified version of NGAC described in Chapter 6.

With the policy and environment correctly defined we build both into models using Alloy. In order to create the policy model we translate our NGAC inspired policy to Alloy through a series of algorithms defined by this work. To create the environment model we use mappings from entities and rules of the environment to signatures and facts of Alloy. A detailed explanation of this part of the process can be found in Chapter 7.

Once both the policy and environment are modeled with Alloy we can construct assertions to perform conflict and redundancy detection. These assertions are used by Alloy to find counter proofs or reasonably formal arguments for a given claim about the model. More details on this part of the process can be found in Chapter 8.

Chapter 2

Literature Review

2.1 Home Area Network (HAN) Case Study

Zahid et al. [8] describes a case study performed on a home area network (HAN) system. This work strives to validate the HAN specification and description language (SDL) models. The authors developed a novel way to translate from SDL to Promela and developed their own closed model that had no interactions from the environment. Their procedure was to translate the SDL model into Promela and then use Spin to verify the Promela model. The translation began with creating an Intermediate Format (IF) which was a combination between multiple high-level specification languages such as UML. They state that the IF was required due to an inability to directly map data structure from SDL to Promela. Zahid et al. use a tool called if2pml to automatically convert the IF model into Promela. Although the conversion is automatic it converts local variable types from "real" to natural variables. This required manual fixing in order to function correctly with Spin. This work presented a semi-automatic way to translate SDL models to Promela models in the context of home area networks. Although the research area is similar, our work is overcoming a different problem entirely. Instead of just translating between two modeling languages, we are verifying the designed behaviors of IoT devices against user generated policy.

2.2 Modeling of Online Social Networks

Bennet et al. [9] explore and demonstrate how access control policies can be specified, analyzed, and enforced in a simple manner. Bennet et al. propose a model to capture online social network (OSN) features using an Entity-Relationship Diagram and an Attribute-Based Access Control model. Bennet et al. shows how this model can be formalized in the Alloy language. They show that the OSN constraints can be captured adequately. Once the model is formalized to Alloy, Bennet et al. depict how misconfigurations can be automatically detected by the SAT-solver within

Alloy. Lastly they describe how such OSN policies can utilize NIST Policy Machine in order to enforce, manage, and change the policy. Although both their work and ours uses Alloy to formalize a newly designed model, the subject of the model is an OSN. The smart home environment we have modeled is unique and encompasses real world situations which cannot be directly mapped to Bennet's work. The analysis we perform on the smart home policies differ from Bennet's work as we are focused on the verification of the policy rule conflicts and redundancies

2.3 Formal Analysis of Spatio-Temporal RBAC

Another work which utilizes Alloy for formal modeling and analysis is by Toahchoodee et al. [10]. In this work Toahchoodee et al. propose a model of access control which allows permissions to be derived using spacial and temporal attributes. The introduction of spatio-temporal information may lead to conflicts within the access control policy. Toahchoodee et al. show how formal analysis of this new access control model can identify such conflicts. Our works have similarities in the fact that we are both using Alloy to detect policy conflicts. However, the environments we have modeled in Alloy are vastly different. There are unique nuances within the smart home model which pose novel situations that must be accounted for. Another difference is that Toahchoodee's work is modeling a Role-Based Access Control (RBAC) model whereas our work is modeling an attribute based access control model. These two access control models are very distinct.

2.4 Evaluating Access Control Policy Verification Tools

Li et al. [11] propose an initial step for a standard evaluation approach regarding access control policy verification (ACPV) tools. Li et al. develop a set of metrics for analytically evaluating ACPVs and a set of oracles and test cases to empirically check the run-time performance of these ACPVs. This work states that policy verification tools such as ACPT [12], Margrave [13], Mohawk [14], and Mutaver [15] all use different mechanisms, and therefore have different capabilities and performances. Li et al. ask important questions such as "... which tool is suitable to

verify Multi-Level Security (MLS) policies?" In their proposed approach they define the following metrics: safety, separation of duty (SoD), completeness, liveness, model-specific properties, inconsistency detection, real-time response, and detection of redundant rules. Li et al. also developed performance test cases which involved static environments of 15 subjects and 8 objects with a varied number of rules increasing from 15 to 120. The oracles they created were in both natural language and XACML. This work will be invaluable when evaluating our policy verification solution on smart home environments.

2.5 Access Control Policy Tool

Access Control Policy Tool (ACPT) [11, 12] is developed by the National Institute of Standards and Technology (NIST) with the intent of aiding in high security confidence levels for critical IT infrastructure. ACPT was designed to automatically check the syntactic and semantic faults of access control policies before deployment. This tool provides a GUI for improved user experience, property checking of access control models through Symbolic Model Verification (SMV), a test suite generated by NIST's combinatorial testing tool ACTS, and XACML policy generation.

2.6 Margrave

Margrave is an academic access control verification tool [13]. Originally designed to analyze role-based access control policies, Margrave provides a mechanism for translating XACML policies into Multi-terminal Binary Decision Diagrams (MTBDD) as a formalization of the policy. Fisler et al. include change-impact analysis as a key part of their work. This is used to determine differences between policies using their MTBDDs generated from the policy. Calculating the differences between two binary decision diagrams are well defined, but more complicated for multi-terminal variants of these diagrams. Through this work Fisler et al. demonstrate their technique for policy comparison using MTBDDs.

2.7 Policy Conflict Analysis

A work by Charalambides et al. titled *Policy Conflict Analysis for Quality of Service Management* [16] explores policy conflicts in QoS provisioning known as Network Dimension. Although the domain is not related to our work, we felt the policy conflict detection performed by Charalambides et al. is closely related to a major part of our verification process. Charalambides et al. work with modality conflicts from [17] and application specific conflicts described in [18]. Their approach for detecting policy conflicts was to define rules based on the mentioned types of conflicts. This referenced work also provides conflict detection examples, giving concrete and real world representations of their theory. Though our work with conflict detection may have been influenced by Charalambides et al. the context and methodology of our solution is unique compared to their work. An example of our differences is that our work includes a policy translation step from NGAC to Alloy. In addition to we had to define conflict detection predicates that are specific to NGAC.

2.8 Alloy vs. Promela

When it comes to lightweight modeling there are two predominate languages used. One is Promela Language paired with its Linear Temporal Logic (LTL) model checking tool called SPIN. The other is Alloy Language paired with its tool Alloy Analyzer. Though these specification languages are both used for model verification, they excel in different areas. On one hand, Promela allows for concurrent processes which help in modeling complex interactions such as distributed systems. Promela can be used to model communication via synchronous or asynchronous channels. These models can then be checked against LTL properties using the SPIN model checker. This would have been our option if we were to model the network communication within our smart home model. However, we focus primarily on access control which is applied to a smart home environment. In regards to Alloy, the use cases are tailored towards expressing structural constraints. Alloy provides a modeling tool based on first-order logic which can be utilized to create "micro-models". These models can then be automatically checked for correctness using the

Alloy Analyzer tool. In this work we decided on the formal modeling language Alloy. Referencing work from Pamala Zave from AT&T Laboratories [19] there are additional general advantages of using Alloy over Promela. These advantages are: faster startup time, safety assertions are declarative, can be used for convincing proof of correctness, and automated visualization. The stated visualization was a very useful aspect that increased our understanding of the model as we developed it. Another key difference between Promela and Alloy is that they excel in state exploration and logical exploration respectively. As the context for this work is not stateful, we found that analyzing the logic of the policy would be more suited as verification over analyzing the state space of our environment.

Chapter 3

Current State of Access Control

3.1 Network-Based

For this section we focus on Network Access Control (NAC) systems that are targeted towards IoT devices and environments. One such vendor is FORTINET. They have a product called FortiNAC [20] used to provide network access control to healthcare devices. These devices are in the space of IoMT or Internet of Medical Things. One issue we find with NAC is the lack of granularity regarding the policies. A majority of these NAC solutions are operating on a port-based schema defined by IEEE [21]. The benefit of using an attribute-based schema is the variety of dynamic situations that can be considered regarding access.

Sivaraman et al. [22] propose a hybrid solution to IoT access control that involves device level protection augmented with network level security solutions. In their work they illustrate the need for IoT access control through an illustration of current smart home threats. The authors motivate the need for having network level control over IoT devices. Some of the benefits include covering a variety of devices, ability to patch and maintain network level solutions, and the inclusion of a third party who has expertise in security. For their prototype they have a traditional home network including a gateway and various smart devices. Attached to the gateway is a OpenVSwitch controller which feeds collected network traffic from the home to an internet service provider through a tool called OpenFlow. Sivaraman et al. propose a three party control architecture which enables a specialist provider to offer security-as-a-service. Although their solution involves network security and privacy control in smart homes, it falls under implementation. Our work on the other hand deals with the verification of the underlying access control policies being implemented.

3.2 Functionality-Based

There have been many access control schema proposed over the years, such as Role-based Access Control (RBAC) [23], Capabilities-based Access Control (CapBAC) [24], and Attribute Based Access Control (ABAC) [25]. Lee et al. [26] propose a unique schema called Functionality-centric Access Control for IoT Programming Frameworks or FACT. Their work is focused on improving two existing flaws in current access control programming frameworks such as SmartThings and IoTivity. These flaws are stated as (1) course-grain access control and (2) lack of resource isolation. They also built a prototype of FACT on IoTivity. This work opposes device-centric models which allow account-to-device and device-to-device authentication. In their work they propose a fine-grained access control system for IoT frameworks. It is the manufacturers responsibility to add this functionality into the IoT device firmware.

3.3 Attribute-Based Access Control

Attribute Based Access Control (ABAC) is a type of access control that makes decisions based on attributes, as opposed to decisions based on the identity or role of a user which is commonly seen in other types of access control. The main elements of ABAC are as follows: subjects, objects, operations, environments, attributes, and policies. These elements and how they are used make ABAC granular and domain independent.

Subjects are entities that can issue access requests to perform an operation on an object. These can be human users or some entities acting on behalf of human users. Objects, sometimes called resources, are entities that have operations performed on them by subjects. Operations are functions that are executed as requested by a subject. These can be actions such as read, write, edit, or delete. Environments are independent of subjects and objects, but they can be used as attributes to contribute to access decisions. Examples of environmental conditions can include time and location. Attributes are characteristics of the subjects, objects, or environments. Policies are rules and relationships that determine if access requests should be granted or denied based on subject privileges, objects, and environmental conditions.

ABAC models are created using these elements, and policies are developed in a way that requests are granted based on the attributes of subjects, objects, and environmental conditions. Other types of access control rely on the identity or role of subjects, which requires knowing this identity about every single subject. Contrarily, ABAC is designed so that relationships between each individual subject and object do not need to be specified.

The design of ABAC results in it having benefits over other types of access control. ABAC allows for more flexibility, and that large amounts of subjects and objects can be specified without detailing each of their individual relationships. We can know relationships and permissions by looking at general attributes, rather than knowing everything about an individual entity. This design also allows for new subjects to be added without needing to modify pre-existing rules and objects. Overall, this makes the models easier to manage and maintain. ABAC also allows access control to be managed in a domain-independent manner and offers fine-grained access control. Despite ABAC's various benefits, it still has some drawbacks. One of the main drawbacks is the cost of implementing ABAC models. Due to the complexity of ABAC systems, they end up being time-consuming to develop and require continual maintenance due to attribute updates.

3.4 XACML

eXtensible access conctrol markup language (XACML) [6] is another attribute based access control (ABAC) framework that is based on Extensible Markup Language (XML) Being a form of ABAC, XACML includes subjects, actions, resources, and environments. XACML uses the term resource, but this is synonymous with the term objects in ABAC and NGAC. XACML is designed to express security policies, access requests, and responses from making policy decisions. These are expressed by using the attributes of subjects, resources, and environments. Attributes in XACML are expressed as name-value pairs. For example, we could define a name as "Role," and the value for that could be "professor," giving us a pair that looks like: Role = "professor."

XACML access policies are defined by a nested structure of PolicySets, Policies, and Rules. The overall structure is a PolicySet, which is composed of Policies, and Policies are composed

of Rules. Rules are defined as boolean conditions that will either permit or deny if the rule is evaluated to be true. A Policy is an algorithm that combines these stated rules, and a PolicySet is an algorithm that combines policies. XACML also introduces the concept of a Target, which is a boolean that, if evaluated to true, will request for a policy decision to be made. Targets are a part of both Policies and PolicySets, where the boolean is a criteria for the Policy or PolicySet to be applicable in a given situation. If the Target is satisfied, then the Policy or PolicySet will be evaluated.

Because of XACML's PolicySet and Policy setup, policies could consist of multiple rules or PolicySets could consist of multiple policies that will evaluate to different decisions. So, XACML accounts for this with the ability to combine algorithms for decision priority. Examples of this are if one Rule permits a request, then the overall decision is permit even if other Rules in the policy return a deny. Likewise, if one Rule denies, then the overall decision can be specified to be deny.

3.5 NGAC

Next Generation Access Control (NGAC) [6] is an ABAC framework developed by NIST. NGAC's goal is to allow for a unified way for diverse policies to be expressed and enforced in the same way. ABAC allows for easier creation and management of policies. Attributes refer to descriptors used for any entities in the policy, like subjects and objects. Attribute based access control has been shown to be effective in providing granular access control IoT devices [25]. NGAC is also useful in modeling state-based policies, as other types of access control do not allow for this.

NGAC differs from other types of attribute-based access controls through its setup. NGAC uses the terms subject, operation, object, and policy class. Entities that have operations performed on them are referred to as objects. These are entities that need protection; like files, records, or devices. Operations in NGAC are input and output operations, like reading or writing, that can be performed on the objects. The policy class is a way to group and organize policy data. So, policy classes are used for grouping subjects, objects, and attributes with specific access control policies. NGAC requires that each subject, subject attribute, object, and object attribute must be contained

in at least one policy class. One aspect of NGAC which separates it from XACML is the concept of obligations. This structure is a way for the policy to alter itself based on whether conditions are met. For example, a policy can update the permissions of a subject based on temporal attributes automatically to create a time dependant policy. Although this is a nice feature of NGAC, we do not use obligations in this work as they are more applicable to stateful policies. That being said, we opted to use NGAC for this work instead of XACML for a couple key reasons. The rules defined for our smart home environment are relatively simple and there is an intuitive process for mapping device specifications to policy attributes. Obligations can be utilized for modeling stateful policies which is one of our future works.

Chapter 4

Background

4.1 NGAC Adapted to IoT

In NGAC, attributes and policy classes are treated as containers. Every entity has its own attributes that are also shared among other entities. For example, we could have Person1 and Person2 who both fall under the attribute category of "Student." In this case, Person1 and Person2 would be subjects and Student would be the attribute.

Though our policy is heavily influenced by NGAC, we modified the framework to remove ambiguity and remove aspects which were not applicable to this application. In this work we modify the Assignment, Association and Prohibition structures. If unfamiliar with NGAC structures, there are three structures that this work is inspired by that make up a policy. These structures are: *Assignments*, *Associations*, and *Prohibitions*. A policy is defined by the combination of these structures. This aspect makes NGAC unique from other types of access control. The three aforementioned structures are essential to creating our smart home policies.

We use Assignments to assign attributes to entities, or to denote membership to containers. Assignments are denoted by *(x, y)*, which means x is assigned to y. Assignments always imply containment, so *(x, y)* would mean that x is contained in y. Assignments are also used to relate attributes through containment relations. Assignment relation are transitive.

Associations are how privileges are derived. These access rights are acquired through association. We express Associations in the form of a 4-tuple depicted below. The tuple contains a subject attribute which is usually a superset of subject attributes and subjects. A access right set which is a set of allowed access rights. An object attribute which is usually a superset of object attributes and objects. Finally an environment attribute.

(subjectAttribue, accessRightSet, objectAttribute, environmentAttribute)

Prohibitions are defined similarly to associations, in the form of a 4-tuple depicted below. The only difference between the association and prohibition, is that associations express granted access and prohibitions express denied access.

deny(subjectAttribute, accessRightSet, objectAttribute, environmentAttribute)

Note that in this work our example policy does not include any environmental constraints, therefore in later discussions of associations and prohibitions we exclude the environmentAttribute from their definitions.

Chapter 5

Modeling IoT Environment

The basis for an IoT environment is defined by the entities contained within the environment and their communication relationships between each entity. In this chapter we formally define the entities considered in our concrete example.

5.1 Smart Home Environment

In smart home environments there are entities which communicate with each other and interact with the physical world. For our model these entities are devices which have had their behavior observed and documented. We have selected few different devices based on their varying behaviors.

5.1.1 Entities

The devices we used in our concrete examples are listed in Table 5.1. This table describes each device and their aspects which will later be organized into our policy attributes.

Table 5.1: Smart-Home Modeled Devices

ID	Device	Type	Manufacturer	Group
Light001	LB100	Light Bulb	TPLink	Utility
Phone001	Nexus5x	Mobile Phone	LG	Controlling, Entertainment
Phone002	Pixel3	Mobile Phone	Google	Controlling, Entertainment
Dimmer001	DimmerSwitch	Light Switch	iDevice	Utility
Camera001	SmartCamera	Security Camera	Arlo	Security, Utility
TV001	SmartTV	TV	Samsung	Entertainment

We have chosen these devices specifically to show variability within a realistic smart home environment. The list of devices is as follows:

Devices:(Light001, Phone001, Phone002,Dimmer001, Camera001, TV001)

5.1.2 Component Sets

Within smart home environments there are sets into which entities are categorized. These sets are not necessarily disjoint and may overlap with multiple sets. This section defines the various entity categorization sets found in smart home environments.

Model: Each device has a model associated with it. We are assuming in our work that there is one instance of a device for each model creating a one-to-one relation between model and device. The enumeration of device models is as follows:

Models:(LB100, Nexus5x, Pixel3, DimmerSwitch, Q, TU8000)

Maker: This is a set of producers for the environment entities. This is a one-to-many relationship where each entity is mapped to exactly one Maker as in a smart home environment a specific device cannot be produced by two manufacturers at the same time, but each maker may have produced many entities.

Makers:(LG, TPLink, iDevice, Samsung, Arlo)

Group: This set collects entities which have commonality of use. This is a many-to-many relationship as an entity may belong to many groups and groups may represent many entities. In other words, devices which are used in the smart home for similar purposes are collected together by a Group set. Examples of a Group are

Groups:(Utility, Entertainment, Control, Security)

Type: This set describes devices which have commonality of function. This is similar to the Group set, but these devices are collected based on their technical functionality opposed to their subjective use. Also similar to the Group set, this is a many-to-many relationship as an entity may belong to many types and groups may represent many entities. This set is more powerful than the Group set and therefore more constrained. Inverse to the Group set, entities of different groups are likely to be in one Type set.

Types:(Light, Controller, Survey, Display)

Chapter 6

Modeling SmartHome Policy

6.1 Smart Home NGAC Policy

6.1.1 Rule set

This work defines 8 simple rules that provide full coverage of our rule archetypes found in Table 6.1. These archetypes are the fundamental pieces in which we can derive rules in this environment. We have taken these archetypes and derived 8 example rules that cover each of these rule fundamentals. These example rules can be found in Table 6.2 and we will refer a set of the first five rules as our *simple rules*. These rules will be the basis for our NGAC policy and will define the following structures.

Table 6.1: Rule Archetypes

1	Allowance of communication from a single device to another single device
2	Restriction of communication from a single device to another single device
3	Allowance of communication from one device to a group of devices
4	Restriction of communication from one device to a group of devices
5	Allowance of communication from one group of devices to a single device
6	Restriction of communication from one group of devices to a single device
7	Allowance of communication from one group of devices to another group of devices
8	Restriction of communication from one group of devices to another group of devices

6.1.2 Subject/Object Attributes

Referring back to our environment model each device has a component set which describes aspects of that device. Each of these aspects are translated into attributes. The following is the

Table 6.2: Comprehensive Policy Rule Set

Rule 1	TPLink Lights may not communicate with Pixel3
Rule 2	Entertainment devices may not send communication to Light001
Rule 3	Utility devices may only communicate with other Utility devices
Rule 4	Lighting devices may communicate with Nexus 5 mobile device
Rule 5	TPLink Light may receive communication if Nexus 5 mobile is connected
Rule 6	Phone001 can communicate with TV001
Rule 7	Phone002 may not communicate with security devices
Rule 8	Dimmer001 may not communicate with Light001

Table 6.3: Rule Set Context

Rule 1	We want restrictions on what devices can control other devices. E.g, a parent may not want their child's phone to be able to control the home lighting, so the child's phone would not be allowed communication with lighting devices.
Rule 2	We don't want entertainment devices, like a TV, having the ability to control lighting and other similar devices.
Rule 3	If we want to use the dimmer, the dimmer must be able to turn the lights on in order to dim them.
Rule 4	We don't want the lighting devices to communicate with random other devices (like TVs or cameras), but we do want to send input to the lights and receive verification of input on the main controlling phone.
Rule 5	Demonstration of conditional restriction between Model attribute and All. Imagine a situation where there are two controllers in the environment. This rule requires both controllers to be present for a specific device to be operated.
Rule 6	Want to control individual devices using the main controlling device.
Rule 7	The second controlling device should not be able to monitor and communicate with any security device or camera.
Rule 8	We don't want a dimmer to communicate with a specific lighting device.

enumeration of attributes for our concrete model. Additional attributes may be created to represent environmental variables such as time, location, or other constraints determined by the environment.

Attributes:{

Model:(LB100, Nexus5x, Pixel3, DimmerSwitch, Q, TU8000)

Maker:(TPLink, LG, Google,iDevice,Arlo, Samsung)

Group:(Utility,Control, Control, Entertainment, Entertainment, Security)

Type:(Lighting, Controller, Controller, Lighting ,Display, Surveillance)}

6.1.3 Assignments and Associations in context

Assignments are used in NGAC to describe relationships between attributes and subjects/objects or between pair of attributes. The following assignments are an NGAC representation of entity descriptors for our concrete example.

Assignments:{

***Model**:(Light001,LB100), (Phone001,Nexus5x), (Phone002,Pixel3), (Dimmer001,DimmerSwitch), (Camera001,Q), (TV001,TU8000)*

***Maker**:(LB100,TPLink), (Nexus5x,LG), (Pixel3,Google), (DimmerSwitch,iDevice), (Q,Arlo), (TU8000,Samsung)*

***Group**:(LB100,Utility), (Nexus5x,Control), (Pixel3,Control), (DimmerSwitch,Entertainment), (TU8000,Entertainment), (Q,Security)*

***Type**:(LB100,Lighting), (Nexus5x,Controller), (Pixel3,Controller), (DimmerSwitch,Lighting), (TU8000,Display),*

(Q,Surveillance)}

Associations define the actions that can be taken between subject attributes and object attributes. To represent the subject and object attributes we use the form "device.X = Y" where X is the assignment label and Y is the value of that label. For example above we see that the device

Phone001 is assigned to the model Nexus5x. In our dot notation this would be Phone001.Model = Nexus5x. An example of an association would be if a particular rule stated that subjects that are assigned to the attribute "device.Maker = TPLink" may only send information to objects assigned the attribute "device.Maker = Samsung", then this would be represented by an association between the subject attribute "device.Maker = TPLink" and the object attribute "device.Maker = Samsung". Because the rule states that the subject may only receive information, the association would take the form {"device.Maker = TPLink", {r}, "device.Maker = Samsung", none}. This is the most basic way of defining access within the our NGAC inspired framework. Notice that there are four elements to the association. The first element is a subject attribute, the second is a set of access rights, the third is an object attribute, and the fourth is a condition. The following defines the associations within our concrete policy:

Associations:{(device.Group = Utility, S,R, device.Group = Utility, None), (device.Type = Lighting, {S,R}, device.Model = Nexus5x, None)}

6.1.4 Prohibitions

Prohibitions in the NGAC framework are a mechanism for defining denials in NGAC. They take a similar form to associations, as both represent a relationship between attributes, though they represent denial of the access rights instead of allowance prohibitions are also between attributes. Prohibitions come in three forms, subject attribute deny (sa_deny), subject deny (s_deny), and process deny (p_deny). However, in this environment only subject and subject attribute denies are used as there is no notion of a process. We condense these two denial types int one form as our concrete example shows below:

Prohibitions:{deny(device.Group = Entertainment, {S}, device.Maker = TPLink, None),
deny(device.Maker = TPLink, {S}, device.Model = Pixel3, None),
deny(device.Maker = TPLink, {R}, device.all, Nexus5x.connected = False)}

In our framework, prohibitions are opposite of associations. Where a associations represent allowance rules, prohibitions represent denials. .

Chapter 7

NGAC Translation to Alloy

7.1 Smart home Environment in Alloy

7.1.1 Device

Each device in this Alloy model is a signature containing identifiers about the entity mentioned in Chapter 5. Shown in Figure 7.1 we model each of the device attributes with context about the real world constraints. Such a constraint is that each specific device is only produced by one manufacturer. For this reason we have the set maker only containing one Manufacturer. A similar reasoning is used when representing the model of said device. We also added a connected which is used in our 5th example policy rule.

```
// Define Devices Blueprint
abstract sig Device {
        model: one Model,
        id: one ID,
        maker: one Manufacturer,
        group: set Group,
        type: set Type,
        connected: one Boolean
}
```

Figure 7.1: Alloy Representation of a IoT Device

7.1.2 Facts

Every smart home has some underlying properties that must be explicitly defined. One example of such properties is the relationship between devices and their manufacturer, group(s), type(s), and connected status. These relationships are not decided by the user, but are inherent to our physical world. One such relationship may be that the LB100 Lightbulb is manufactured by TPLink. This is a "fact" that has always been and will always be true in this system. In Figures 7.2 and 7.3 we show the facts of our example smart home defined through Alloy.

```
//Mapping between device and maker
//each device has one maker, this fact relates devices
//to makers
fact {maker = Light -> TPLink +
            Phone1 -> Google +
            Phone2 -> LG +
            Dimmer -> iDevice +
            Camera -> Arlo +
            TV -> Samsung
}

//Mapping between device and group
//each device belongs to on or more groups
fact{ group = Light -> Utility+
            Phone1 -> Control +
            Phone1 -> Entertainment +
            Phone2 -> Control +
            Phone2 -> Entertainment +
            Dimmer -> Utility +
            Camera -> Security +
            TV -> Entertainment
}
```

Figure 7.2: Alloy Smarthome Facts

7.2 Modeling Rules

We represent allowance and denial rules in NGAC through associations and prohibitions. These two structures define sets of states that are either allowed or denied respectively. A state in this context is a unique configuration of source and destination devices from the modeled environment. The set of all prohibitions in a policy define the "bad states" of the environment. Bad states in this context are any state that the policy author has specified that must be avoided. Conversely, associations define a subset of "good states" for the policy. Good states in this context are any situation which the author either specifies as allowed, or unspecified states.

In Alloy we use the same structure to represent both associations and prohibitions. The predicate structure in Alloy allows us to define boolean expression which can be checked with the Alloy Analyzer for any states that satisfy this predicate. Although our representation of associations and prohibitions in Alloy are syntactically similar, the semantic differences is what really defines them. Syntactically each predicate is a boolean AND between various descriptors of the source and destination devices. Using this technique we can logically represent the access situation described

```
//Mapping between device and type
//each device belongs to one or more types
fact{ type = Light -> Lighting +
            Phone1 -> Controller +
            Phone2 -> Controller +
            Dimmer -> Lighting +
            Camera -> Survey +
            TV -> Display
}

//Mapping between Device and Model
fact{model = Light -> LB100 +
            Phone1 -> Pixel3 +
            Phone2 -> Nexus5x +
            Dimmer -> Dimmer_Switch +
            Camera -> Q +
            TV -> TU8000
}

//Mapping between Device and ID
fact{id = Light -> Light001 +
      Phone1 -> Phone001 +
      Phone2 -> Phone002 +
      Dimmer -> Dimmer001 +
      Camera -> Camera001 +
      TV -> TV001
}
```

Figure 7.3: Alloy Smarthome Facts Cont.

by the prohibition or association. This can be seen in Figures 7.4 - 7.8. Semantically, the predicates that represent prohibitions are logically describing a "bad state". Therefore, if the predicate evaluates to True, then said "bad state" has occurred and access should be denied. If the predicate evaluates to False, then this means that the bad state has not occurred. We see this in Figure 7.4 where the predicate is modeling the two situations where: 1) the source device is manufactured by TPLink and the destination device model is Pixel3, and 2) the source device model is Pixel3 and

```
//Rule: TPLink Light may not communicate with Pixel 3
//Context: We want restrictions on what devices can control other devices. E.g, a parent may not want their child's phone to
//            be able to control the home lighting, so the child's phone would not be allowed communication with lighting devices.
pred noTPLinkToPixel3[src, dst: Device] {
 (src.maker = TPLink) && (dst.model =  Pixel3) || (src.model = Pixel3) && (dst.maker = TPLink)
}
```

Figure 7.4: Simple Rule 1 as Predicate

This rule models the bad states where the source device is manufactured by TPLink and the destination device is of model Pixel3. Or the source device is of model Pixel3 and the destination device is manufactured by TPLink.

```
//Rule: Entertainment devices may not send communication to Light001
//Context: We don't want entertainment devices, like a TV, having the ability to control lighting and other similar devices
pred noEntertainToTPLink[src, dst: Device] {
   (Entertainment in src.group) && (dst.id = Light001)
}
```

Figure 7.5: Simple Rule 2 as Predicate

This rule models the bad states where the source device has Entertainment as one of its group attributes and the destination device has an ID of Light001

```
//Rule: Utility devices may only communicate with other Utility devices
//Context: ex: If we want to use the dimmer, the dimmer must be able to turn the lights on in order to dim them
pred utilityComm[src, dst: Device] {
   (Utility in src.group) && (Utility not in dst.group)
}
```

Figure 7.6: Simple Rule 3 as Predicate

This rule bad states the situations where the source device has Utility as one of its group attributes and the destination device does not.

```
//Rule: Lighting devices may communicate with Nexus 5 mobile device
//Context: We don't want the lighting devices to communicate with random other devices (like TVs or cameras), but we do
//want to send input to the lights and receive verification of input on the main controlling phone
pred lightingComm[src, dst: Device] {
   (Lighting in src.type) && (dst.model = Nexus5x)
}
```

Figure 7.7: Simple Rule 4 as Predicate

This rule models the bad states where the source device has Lighting as one of its type attributes and the destination device model is Nexus5x.

```
//Rule: TPLink Light may receive communication if Nexus 5 mobile is connected
//Context: We don't want lighting or other home devices to be controlled when the main homeowner is not home
//E.g, we don't want a child controlling the home lights if the parent/parent's device are not home
pred DenyIfNexusNotConnected[src,dst: Device]  {
(dst.model = LB100)  && (src.model = Nexus5x) && (src.connected = False)
}
```

Figure 7.8: Simple Rule 5 as Predicate

This rule models the bad states where the destination device model is LB100 and the source model is Nexus5x and the source device is not connected.

the destination device manufacturer is TPLink. If this predicate returns True, then either of these two situations have occurred. Being that this predicate is representing a prohibition, access should be denied.

On the other hand, predicates that represent associations are logically describing a "good state". Therefore, if the predicate evaluates to True, then said "good state" has occurred and access should be granted. If the predicate evaluates to False, then this means the specific "good state" represented by the predicate has not occurred. In the case of an association a False evaluation is not necessarily a negative result. As we see in Figure 7.7 the predicate is defining the situation where the source device type contains Lighting and the destination device model is Nexus5x. If this predicate were to evaluate to True, then we know that a "good state" has occurred and access should be granted.

Note that due to syntactic similarity between the predicates representing prohibitions and associations it can become confusing which semantic interpretation should be used. To mitigate this confusion a naming convention can be used to clearly differentiate predicates representing prohibitions and predicates representing associations. It is essential to be able to differentiate between these predicates as they will be integral to rule conflict and redundancy identification later in Chapter 7.

7.3 Privilege and Restriction Derivation

The next step in the process for translating NGAC to Alloy requires us to take the allowed and denied situations gathered from the policy, represented as privileges and restrictions, and convert them into Alloy syntax. A privilege or restriction is an atomic structure within NGAC that defines one covered situation. Shown below we represent this artifact as a tuple. This tuple describes the subject, access right, and object involved in the privilege or restriction. The syntax of privileges and restrictions are similar in form but as we discuss in this section there is a fundamental semantic difference between the two.

$$(subject, accessright, object) \tag{7.1}$$

Privileges define situations of access that are allowed by the policy. Algorithm 1 demonstrates how to generate all privileges for a given policy. This is done by first stepping through all associations in the policy. These associations take the form shown below.

$(subjectAttribute, accessRightSet, objectAttribute)$

which needs to be converted to a set of privileges. For each association we follow the subject and object attributes up to their leaves which are of type subject and object respectively. Associations define a set of access rights between the two different attributes. Once we have a set of subjects, a set of objects and a set of access rights, we can create a Cartesian product of all three of these sets. This gives us all possible permutations of subject, object, and access right. However, these privileges only give us a representation of what is explicitly allowed in the NGAC policy.

Restrictions define the situations of access that are not allowed by the policy. The algorithm designed to define denials in the NGAC policy is the same as Algorithm 1 but we use the prohibitions set instead of the associations set. The algorithm steps through each prohibition enumerating all restriction tuples.

```
Result: All Policy Privileges are defined
for each association in Associations do
    Perform DFS to create set of all subjects indirectly assigned to
     association.subjectAttribute;
    Perform DFS to create set of all objects indirectly assigned to
     association.objectAttribute;
    for accessRight in accessRight Set do
        for subject in new subject set do
            for object in new object set do
                if subject does not equal object then
                    save (subject, accessRight, object)
                else
                    pass
                end
            end
        end
    end
end
```

Algorithm 1: Privilege Definition for a Policy

7.3.1 Privileges in Alloy

A simple allowance is a single privilege representing a situation where a subject is accessing an object through an access right. This relation can be expressed in allow with a simple logical expression. For example, given the following:

(Light001,R,Phone001)

This states that Light001 is allowed to receive information from Phone001. The situation describes that if a device with the identifier of Light001 is receiving information, then the sender is allowed to be Phone001. We can represent this using a logical AND where the receiving (destination) device ID is Light001 and the sending (source) device ID is Phone001 as depicted below.

(dst.ID == Light001) && (src.ID == Phone001)

In the context of our Smart home environment, the predicate for allowances are depicted in Figures 7.7 and 7.8.

7.3.2 Restriction in Alloy

A simple denial is a single restriction which represents a single situation where a subject is explicitly rejected from accessing an object through a specified access right. Similar to simple allowances, this can be represented with the logical form of two or more clauses ANDed together.

Given the denial:

deny(Light001,R,Phone001)

We verbalize this situation as Light001 is not allowed to receive information from Phone001. This is represented as the clause source device ID is Phone001 AND the clause destination ID is Light001. This expression is shown below:

(dst.ID == Light001) && (src.ID == Phone001)

Notice how the expression for the privilege is exactly the same as the expression for the restriction. This is due to the given association and prohibition are composed of the same structure. Syntactically these two structures are similar, but semantically they are opposites. We use this fact in the conflict identification section of Chapter 8.

The predicates for our Smart home environment regarding denials are depicted in Figures 7.4 - 7.6.

Chapter 8

Policy Analysis with Alloy

Our utilization of NGAC creates opportunities for policy conflicts and redundancies to arise. This section provides details for these situations as well as workflows for identifying their occurrences. These workflows take an incremental approach where additional associations and prohibitions are compared against a pre-defined policy. For redundancy and conflict situations we start with a manual technique of identification. After this we show the benefit of simplicity when using Alloy to achieve the same results. We also note that Alloy provides additional functionality by specifically identifying redundant or conflicting rules rather than just their existence.

8.1 Redundancy Identification

Problem Setting. In this subsection we consider the scenario where a policy P is defined and a new rule r is being added to the policy. Once the rule has been added we compare the new policy $P + r$ with the original policy.

*A **redundancy** occurs when an association or prohibition is subsumed by another association or prohibition respectively*. This is expressed by the addition of one of these structures to the policy resulting in no change in privileges for associations or no change in restrictions for prohibitions.

Redundancy between prohibitions occur when one prohibitions scope overlaps another. This is often recognized as a broad prohibition subsuming a more narrow prohibition. An example of this would be a broad prohibition stating that "TPLink manufactured devices are denied communication to Google manufactured devices" and a more narrow prohibition stating "Light001 is denied communication with Phone002". These may seem like distinct prohibitions, but Light001 is in fact a TPLink manufactured device. Similarly, Phone002 is a Google manufactured device. The set of situations covered by the narrow prohibition is contained within the set of situations created by the

broad prohibition. Therefore if we remove the narrow prohibition the restrictions defined by the policy will not change as the narrow prohibition is not restricting any unique situations.

Here we setup a workflow for identifying prohibition redundancies as mentioned in this section. In order to identify this prohibition redundancy we take into account:

P: The access control policy
ORS: The set of restrictions for the given policy P (defined in Chapter 7, restriction is a single situation where the elements of the restriction are explicitly denied access)
OPS: The set of privileges for the given policy P (defined in Chapter 7, privilege is a single situation where the elements of the privilege are explicitly allowed access)
S: The set of restrictions derived from the new rule r

The following workflow is in terms of prohibitions but due to the syntactic similarity associations can be interchanged. To identify a redundant association added to the policy, the restriction set of P is exchanged for the privilege set of P. For this case we would be identifying the subsumption of a new association by the existing policy associations. In general, we perform a comparison of the restriction set of P before a new rule r is added and the restriction set after the rule has been added to P. This means each element in ORS is a tuple of the form: $(Subject, AccessRight, Object)$. Next we derive the restriction set of the to be added prohibition r. This set describes the situations which this prohibition is denying access. Next we add the new prohibition to the policy and derive the new restriction set NRS:

$$NRS = ORS \cup S \tag{8.1}$$

Now that we have ORS which is a set of all restrictions before the new rule was added to P, and we have the NRS which is a set of all restrictions after the new rule has been added, we can perform set operations on them to identify the existence of redundancies. The goal of this opera-

tion is to determine whether NRS is a strict superset of ORS. The operation is as follows:

$$NRS \setminus ORS \tag{8.2}$$

If the rule that was added is a redundant rule, then the resulting set of this calculation would be the empty set. If the result is not the empty set then this tells us the prohibition is not redundant and has provided additional restrictions to the policy. Therefore the existence of a redundancy, whether that be a prohibition or association, with respect to the policy P can be identified with just set-algorithms.

Rule Redundancy Identification Using Alloy

As mentioned we can identify the existence of redundancies through manipulation of policy derived sets. However, this method does not allow us to identify which rules specifically are redundant within the policy. Along with this, the process of deriving sets and executing Algorithm1 can be tedious and susceptible to errors. This is where Alloy comes in. Through modeling the policy in Alloy we are able to perform the same functionality as with set manipulation, but we gain increased visibility to which rules are flagging the redundancy. This also reduces the process complexity as once the policy is modeled and assertions created, Alloy does the rest of the heavy lifting.

The strategy we take in Alloy is to identify whether an added rule is redundant with respect to an already defined policy. We create a redundancy situation in Alloy where an added association is redundant with respect to the existing policy. The new association to be added is found in Figure 8.2 and the existing policy association is labeled "allowance" in Figure 8.6. With the added rule and existing policy defined in Alloy the natural course is for us to create an assertion to represent the situation where the new rule is redundant. This assertion is depicted in Figure 8.1. This assertion highlights situations where the "new_allowance" is not redundant. However, redundancy is not the normal case, more often than not we expect for an added association to be impact to the

policy. Therefore we want to assert that the new rule is not redundant. Our goal is to have Alloy return true when there is no redundancy, and alert us with a counter proof when there is a redundancy. Due to this perspective we must negate the previously mentioned redundancy assertion. The negated assertion is shown in Figure 8.3. This negated redundancy assertion states that there exists source and destination devices, such that the "new_allowance" predicate is satisfied and the "allowance" predicate is not satisfied. This follows our status quo as if both "new_allowance" and "allowance" were satisfied for all source and destination devices then "new_allowance" is redundant. In this example we see in Figure 8.4 that Alloy does indeed catches this redundancy by providing a counter proof that the situations allowed by the new_allowance predicate is a subset of the situations allowed by the policy allowance predicates. This proof states that for all possible source and destination devices satisfaction of the new rule, new_allowance, implies satisfaction of the policy allowances.

Note, in our demonstrative example we use only two predicates. to match a more likely scenario with more than one association predicate in the policy we simply group OR the association predicates defined by the policy, similar to what is seen in Figure 8.9.

```
assert redundant{
      all src, dst: Device | new_allowance[src, dst] => (allowance[src, dst])
}

check redundant for 5
```

Figure 8.1: Redundancy Assertion

So far we have shown equal functionality to the previous technique where we are detecting the existence of a redundancy. However, through Alloy Analyzer we are given generated visualization and report of the counter proof. This output describes the exact situation which violated the assertion. We can now take this counter proof situation and cross reference it with the policy allowances in order to arrive at the specific association predicates the new association is redundant to.

```
pred new_allowance[src, dst: Device]{
    (src.id = Light001) && (dst.model = Pixel3)
}
```

Figure 8.2: Redundant Allowance Predicate

modeling the good state that the source device has ID of Light001 and the destination device with model Pixel3.

```
assert notRedundant{
    some src, dst: Device | new_allowance[src, dst] && not (allowance[src, dst])
}

check notRedundant for 5
```

Figure 8.3: Negated Redundancy Assertion

stating that there exists some source and destination device such that the new_allowance predicate is satisfied and the allowance predicate is not satisfied.

8.2 Complete Rule Conflicts

Problem Setting. In this subsection, much like the previous, we consider a scenario where a policy P is defined and a new rule r is being added to the policy. We then compare the new policy $P + r$ with the original policy.

*A **conflict** occurs when the privileges introduced by an association are overruled by restrictions derived from the policy where an assertion states an allowance and a prohibition states the opposite (this is known as a modality conflict).* An example of this would be a policy stating that "Device X *can not* communicate with Device Y" and adding a rule to the policy stating that "Device X *can* communicate with Device Y". These two rules are stating that opposite situations must be satisfied at the same time, which is logically impossible.

As policy rules are represented as associations or prohibitions in NGAC we describe a conflict in these terms. Modality conflicts in particular occur between associations and prohibitions. This matches the example above where all the elements of each rule are identical except for the terms *can* and *can not*.

Our workflow for conflict identification takes a similar approach to redundancy identification as we are adding a rule r then performing a comparison of the new policy $P + r$ with the original policy P. For this identification we use:

```
Executing "Check notRedundant for 8"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  981 vars. 220 primary vars. 1616 clauses. 20ms.
  Counterexample found. Assertion is invalid. 12ms.
```

Figure 8.4: Redundant Assertion Output

```
assert redundant{
      all src, dst: Device | new_allowance[src, dst] => (allowance[src, dst])
}

check redundant for 5
```

Figure 8.5: Alloy Redundancy Proof

stating that for all source and destination devices if the new_allowance predicate is satisfied then the allowance predicate is always satisfied.

P: The access control policy

ORS: The set of restrictions for the given policy P (defined in Chapter 7, restriction is an single situation where the elements of the restriction are explicitly denied access)

OPS: The set of privileges for the given policy P (defined in Chapter 7, privilege is an single situation where the elements of the privilege are explicitly allowed access)

S: The set of prohibitions in P

A: The set of associations in P

Note that ORS is derived from flattening the prohibition set S where each element of ORS is an atomic situation covered by the set of prohibitions S in policy P. Likewise, OPS is derived from flattening the association set A where each element is an atomic situation covered by the association set A in policy P.

To identify whether an added association r is in conflict with the policy P we first derive the privilege set of r. If this privilege set is a subset of ORS, then we know that r is in conflict with the policy. We know this due to the nature of modality conflicts regarding privileges being the semantic opposite of restrictions. Therefore, if any privilege provided by the added association is represented as a restriction provided by an existing prohibitions within the policy P, we have a conflict.

$$\textit{privilege set of association } r \subseteq ORS \tag{8.3}$$

If the result of this operation is true, then we have identified the existence of a conflict between the added rule r and original policy P. In this case, depending on the policy P the privilege set of r could either be a subset or proper subset of ORS. To be a subset, all privilege elements derived from r are also in ORS. To be a proper subset, ORS must have at least one element not derived from r. If the result of this operation is false then we conclude that the added rule r is not in conflict with the original policy P. Although this set manipulation technique can identify the existence of a conflict between an added rule r and original policy P, we need to utilize Alloy in order to gain vision of which specific prohibition(s) within the policy that r is in conflict with.

Rule Conflict Identification Using Alloy

Much like the previous section on redundancy, here we create a situation to highlight rule conflict identification in Alloy. This conflict is between a new rule stating that "LB100 model device may send to any Pixel3 model devices" and the policy which includes a prohibition that states "TPLink manufactured devices may not send to Google Manufactured devices". As shown in Figure 8.6 these rules refer to different levels of granularity and it is not apparent that these two rules specifically are in conflict. Because we modeled each rule as an predicates in Chapter 5, we can craft an assertion comprised of said predicates in a way that will reveal the conflict if it exists. Naturally we may want to create an assertion to stating that for all source and destination devices, if the allowance predicate is satisfied then the denial predicate must be satisfied. This matches our definition of conflict and equation 8.3. However, much like the redundancy identification, we want the assertion to represent the status quo. With conflicts we expect a new rule to not be in conflict with the existing policy. Therefore we need to negate the assertion which we illustrate in Figure 8.8. With this negated assertion we are declaring the absence of a conflict, and are given a counterexample if there is in fact a conflict. If Alloy finds a counterexample to this negated assertion, meaning there does not exist a combination of source and destination devices such that

proposed allowance rule is satisfied and the policy denial is not satisfied, then we can say that the additional rule is in conflict with the exiting policy. When running this assertion in Alloy we see the output shown in Figure 8.10. From this we see that Alloy has indeed detected a conflict and provided a counter example to our assertion

```
pred allowance[src, dst: Device]{
      (src.model = LB100) && (dst.model =  Pixel3)
}

pred denial[src, dst: Device]{
      (src.maker = TPLink) && (dst.maker = Google)
}
```

Figure 8.6: Rule Conflict Example Rules

```
assert completeConflict{
      all src, dst: Device | allowance[src,dst] => (denial[src,dst])
}

check completeConflict for 5
```

Figure 8.7: Complete Conflict Assertion

stating that for all source and destination devices if the allowance predicate is satisfied then the denial predicate is always satisfied.

Note that this example is for demonstrative purposes only uses association and one prohibition. It is likely that a policy will have multiple prohibitions instantiated at the same time. In this case we group OR all prohibition predicates together as shown in Figure 8.9.

```
assert noCompleteConflict{
      some src, dst: Device | allowance[src,dst] && not (denial[src,dst])
}

check noCompleteConflict for 5
```

Figure 8.8: Negated Complete Conflict Assertion

stating that there exists some source and destination devices such that the allowance predicate is satisfied and the denial predicate is not satisfied.

So far we have shown that the existence of a conflict can be identified using Alloy Analyzer. Though we have not improved the functionality of the previous technique, Alloy Analyzer has a visualizer and report generator for the counterexamples that are discovered. Using this one can then analyze the situation described by the counterexample and identify the source of the conflict by comparing the counterexample situation with the situations covered by each prohibition defined in the policy.

```
assert noCompleteConflict{
      some src, dst: Device | allowance[src,dst] && not (denial[src,dst] || denial2[src,dst] || ... || denialN[src,dst])
}

check noCompleteConflict for 5
```

Figure 8.9: Multiple Denial Conflict Assertion

showing the syntax of having multiple denial predicates in the given policy. The highlight of this figure is the union of denial predicates. This form can be applied to the redundancy assertions as well

```
Executing "Check noCompleteConflict for 5"
  Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
  816 vars. 196 primary vars. 1328 clauses. 113ms.
  Counterexample found. Assertion is invalid. 28ms.
```

Figure 8.10: Concrete Conflict Check Assertion Output

8.3 Partial Rule Conflicts

In the previous section we discussed conflicts where the derived privileges of a new association are being subsumed by the existing restrictions within the policy. However, there is potential for a case where the derived privileges of a new association are only partially subsumed. In this case the associations is still adding something to the policy, therefore we do not consider this a conflict. That being said it may be favorable for the user to be notified that some of the privileges stated by the association are being overridden by the policy restrictions.

Utilizing Alloy we can construct an assertion to identify when this is happening. To begin we define an assertion to model the behavior of this partial conflict shown in Figure 8.11. This assertions states that there exists source and destination devices such that the allowance predicate and denial predicate are satisfied at the same time. However, much like the previous identifications, we want to model the status quo and have the partial conflict be an anomaly. Therefore we must negate this assertion to Figure 8.12. The negated partial conflict assertion states that for all source and destination devices, neither the allowance or denial predicates are satisfied at the same time. In the case where this assertion is violated, we have found a source and destination devices such that the allowance predicate and denial predicate are satisfied at the same time. This matches the original partial conflict assertion where we modeled the situation. However, this information alone is not sufficient to detect the existence of a partial conflict. We pair this test with the conflict identification test mentioned earlier. In the event of a partial conflict we need to ensure that there isn't a full conflict. This is because the situations covered by comlete conflict will also trigger the partial conflict due to the partial conflict assertion checking for at least one conflict between the added rule and existing policy. Therefore, a positive reading for partial conflict occurs when the noCompleteConflict assertions (Figure 8.8) is satisfied while simultaneously the noPartialConflict assertion is not sastisfied. In this case we know we do not have a complete conflict, but there are still some conflict between the added rule and policy.

The same alteration applies to this assertion to account for multiple denials as seen in Figure 8.9.

```
assert partialConflict{
    some src, dst: Device | allowance[src, dst] && denial[src, dst]
}

check partialConflict for 5
```

Figure 8.11: Partial Conflict Assertion

stating that there exists some source and destination devices such that the allowance predicate is satisfied and the denial predicate is satisfied.

```
assert noPartialConflict{
        all src, dst: Device | not (allowance[src, dst] && denial[src, dst])
}

check noPartialConflict for 5
```

Figure 8.12: Negated Partial Conflict Assertion

demonstrating the negation of the Partial Conflict Assertion.

8.4 Alloy Check Bounding

In the verification section we utilize Alloy assertions to check for rule conflicts or property violations. You may have noticed that along with defining assertions we also included a "check" command for a specific assertion. This is the execution command which runs a specific assertion against the modeled universe. Although Alloy can model an infinite universe, it cannot perform model checking on such a universe. Therefore, Alloy requires a maximum depth bound. This is the number after "for" in the check statements. Every assertion has an inherent minimum depth required for a reasonably formal argument. The minimum required depth can be derived by observing how many elements in the universe would need to exist to falsify the assertion, given the assertion could be falsified. Each of our assertions requires at the minimum one source and one destination. Therefore the minimum number of elements in existence for our environment is two. If the assertion was able to be proven false, only one of the variables could be configured in a way to render the assertion false.

8.5 Access Control Checking

Access Control checking is a valuable functionality where a query is provided with respect to a policy and it is determined whether that query is valid or not. In our context this query would be a tuple of the subject, access right, and object. This is the exact format of privileges derived from NGAC. Therefore in order to check whether that query is allowed in the policy we perform this operation:

$$query \cap (privileges/restrictions) \tag{8.4}$$

If the result of this operation is the empty set, then the query is denied by the policy. If the result is the query itself, then we know the access described by the query is allowed by the policy.

Chapter 9

Medical IoT Scenario

In this chapter we walk through a practical scenario involving the preservation of critical device functionality between access control policy versions. Two of the core benefits of this work is conflict and redundancy identification. Specifically conflict identification is useful when altering a policy to ensure such changes do not compromise defined core functionality.

The premise of this scenario is that there exists a smarthome environment with a couple medical devices present. We use smart insulin pumps for this example. Either the home administrator or member of the household requires these insulin pumps to control a diabetic condition. publicly released was a report that the current insulin pump in use, Medtronic's MiniMed 508, is vulnerable to remote compromise. Hearing of this the household decides to transition the operating insulin pump from the MiniMed 508 model to the MiniMed 770G model. As well as adding this new device they decide to restrict access to the MinMed 508 model pump. Following this decision the home administrator alters the access control policy, adding allowance rules for the new MiniMed 770G model device and restrictions for the old MiniMed 508 model device. However, the restriction was mistakenly placed on the manufacturer attribute which consequently halts access to the MiniMed 770G model device due to it sharing the same manufacturer as the MiniMed 508 model device. This restriction does indeed accomplish the administrator's goals in restricting the MiniMed 508, but the administrator may not realize that the MiniMed 770G has also been restricted. Our process for policy analysis through allow can catch such mistakes and therefore notify the administrator that there has been an error in the policy change.

Note: a rational decision would be to disconnect the MiniMed 508 model device from the network altogether, but as we are working in an environment where the administrator may not be experienced or knowledgeable about correct security decisions. As a result irrational actions may occur with good intentions but dangerous repercussions.

The following is a walkthrough of our process in the context of the previously described scenario in order to provide a practical example.

9.1 Define The Environment

The environment for this scenario is a collection of three devices to describe a simplistic home network. Shown in Figure 9.1 we have the description of each device. First we have a mobile phone which is used to control devices within the network. Second, we have two IoT insulin pumps with the Medtronic MiniMed 508 having a known remote access vulnerability.

Table 9.1: Motivating Example Devices

ID	Device	Type	Manufacturer	Group
Phone001	Nexus5x	Mobile Phone	LG	Controlling, Entertainment
Medical001	MiniMed 508	Insulin Pump	Medtronic	Medical
Medical002	MiniMed 770G	Insuline Pump	Medtronic	Medical

9.2 Build The NGAC Policy

In this section we work with only a few rules in order to highlight the specific scenario demonstrating the assurance of functionality for the IoT insulin pump MiniMed 770G withstanding alterations of the policy.

9.2.1 Original Policy

As our scenario environment is compact the rule set for the policy is also compact. The original policy only had one rule stated in Table 9.2 that Medtronic devices may communicate with the controlling phone Nexus5x. This rule is rational as it provides sufficient access and due to the relational nature of NGAC the administrator does not have to alter the policy if another Medtronic device is installed. Although this is only convenient for the administrator if there is no need to alter the policy regarding Medtronic devices. We see later in this chapter the pitfall that can arise from such relational control.

Table 9.2: Original Scenario Policy

Rule 1	Medtronic devices **may** communicate with Nexus5x

9.2.2 Altered Policy

After the administrator had received news that one of their Medtronic devices could pose a security risk they decided to install a new secure device and restrict the old device. In order to restrict the old device the administrator must change the access control policy. Seen in Table 9.3 the administrator made simple yet not obvious mistake when altering the policy. They switched the modality of Rule 1 and added a new rule that explicitly allowed access to the new insulin pump. However, as NGAC is a deny precedence framework a conflict occurs. To the administrator this may not be apparent due to lack of knowledge regarding the NGAC Framework and its deny precedence. We use Alloy in the next sections to detect this conflict.

Table 9.3: Altered Scenario Policy

Rule 1	Medronic device **may not** communicate with Nexus5x
Rule 2	MiniMed 770G may communicate with Nexus5x

9.3 Translate to Alloy

In the previous section we defined the original home policy and the altered home policy. In this section we translate each policy from common language to NGAC, and finally to an Alloy representation which we can perform policy analysis on.

Mentioned in the last section the main issue that arises from this scenarios policy alteration is the fact that NGAC is a deny precedence. This is a common practice for many access control schema though we cannot assume the home administrator will know this information. We demonstrate using NGAC assignment and association graphs to highlight the fault caused by this deny precedence.

9.3.1 Environment Translation

At the beginning of this chapter we defined the scenario environment which included three devices: one phone and two medical devices. Here we carry out the translation process described in detail in Chapter 7.

First we model the component sets of the environment which are represented in NGAC as attributes. Seen in Figure 9.1, we model: Manufacturer, Model, Group, Type, and ID.

Next we create a device "blueprint" or abstract signature that defines a device. This is depicted at the bottom of Figure 9.1. The device "blueprint" defines the structure of a device, incorporating every component set previously modeled.

After this we model the relations between attributes to create NGAC assignments. Seen in Figure 9.2 we model the manufacturers, types, models, etc. of each device.

9.3.2 Original Policy Translation

Shown in Figure 9.3 we see that Rule 1 creates an association between the subject attribute Medtronic and object attribute Nexus5x. As mentioned previously in Chapter 6 allowance rules directly map to associations.

Figure 9.4 depicts how this policy is modeled in Alloy. There is only one predicate in this policy as there is only one rule in the policy.

9.3.3 Altered Policy Translation

The altered policy has had two events take place. The first action was changing Rule 1 to a denial rule instead of an association rule. This in turn changes the NGAC representation of Rule 1 to a prohibition. The second event that occurred was a new Rule 2 was added. This is an allowance rule between MiniMed 770G modeled device and Nexus5x modeled device. As Rule 2 is an allowance this is translated into an association in our NGAC policy. We see the changes when comparing the original assignment and association graph found in Figure 9.3 and the new assignment and association graph found in Figure 9.5.

The combination of these two rules creates a problem. It turns out that Rule 2 is overridden by Rule 1. This is due to the deny precedence where a denial rule will be considered first when queried for a decision. In this case the policy will always return that the MiniMed 770G is denied communication with Nexus5x devices due to Rule 1. Rule 2 is essentially ignored.

Figure 9.6 depicts the altered policy in Alloy. As we have added a new rule there are two predicates in this policy. Notice how the new Rule 1 predicate has not changed syntactically on the inside compared to the original Rule 1 predicate. However, the name has changed due to the change in semantics. Previously the action was allowed, now the action is denied.

9.4 Analyze Altered Policy

In this section we take into account the Alloy representation of the original policy and altered policy in order to identify any conflicts that have been introduced. As our process defines in Chapter 8 we can fit the two events that altered the original policy to the workflow of adding a rule, then checking if it is in conflict with the policy. As the first edit to the original policy was flipping the modality of Rule 1 we will assume this comprises the existing policy. Because Rule 2 is an allowance we add it to the policy and check for conflicts. In this case we are technically checking the addition of Rule 2 against the existence of Rule 1 but as mentioned in Chapter 8 this can be expanded to a policy of many prohibitions.

The assertion created is referenced in Figure 9.7 where we can see that the added Rule 2 is being compared to the policy containing only Rule 1. After running this assertion we see in Figure 9.8 that Alloy does indeed catch that there is a conflict present. With a provided counter example we can do further analysis if needed.

```
//Define NGAC Assignments

fact{
        maker = Nexus5x -> LG +
                MiniMed508 -> Medtronic +
                MiniMed770G -> Medtronic
}

fact{
        model = Phone1 -> Nexus5x +
                Medical1 -> MiniMed508 +
                Medical2 -> MiniMed770G
}


fact{
        group = Nexus5x -> Controlling +
                Nexus5x -> Entertainment +
                MiniMed508 -> Medical +
                MiniMed770G -> Medical
}


fact{
        type = Nexus5x -> Mobile_Phone +
               MiniMed508 ->Insulin_Pump +
               MiniMed770G -> Insulin_Pump
}

fact{
        id = Phone1 -> Phone001 +
             Medical1 -> Medical001 +
             Medical2 -> Medical002
}
```

Figure 9.1: Scenario Component Sets in Alloy

```
//Define Device Manufacturers
abstract sig  Manufacturer {}

one sig LG, Medtronic extends Manufacturer {}


//Define Device Models
abstract sig Model {}

one sig Nexus5x, MiniMed508, MiniMed770G extends Model {}


//Define Device Groups
abstract sig Group {}

one sig Controlling, Entertainment, Medical extends Group {}


//Define Device Types
abstract sig Type {}

one sig Mobile_Phone, Insulin_Pump extends Type {}


//Define Device IDs
abstract sig ID {}

one sig Phone001, Medical001, Medical002 extends ID {}


//Define Device Blueprint
abstract sig Device {
        model: one Model,
        make: one Manufacturer,
        group: set Group,
        type: set Type,
        id: one ID
}

one sig Phone1, Medical1, Medical2 extends Device {}
```

Figure 9.2: Scenario Assignments in Alloy

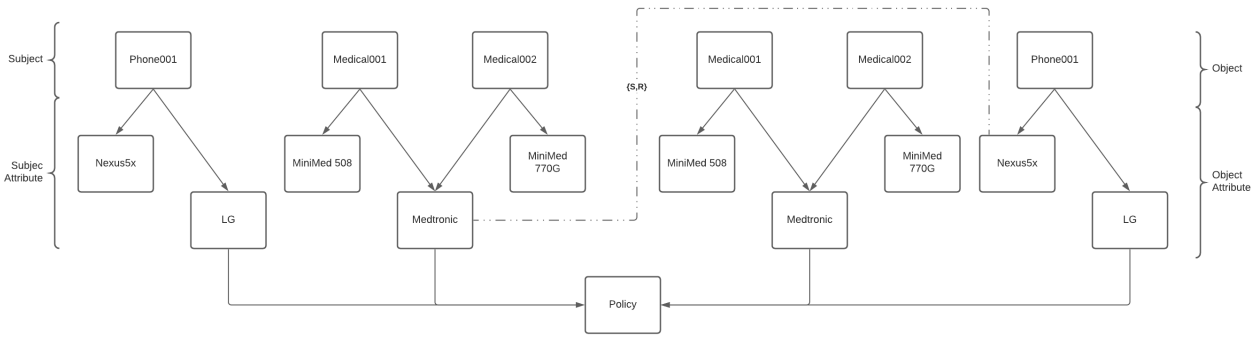

Figure 9.3: Original Policy Assignment & Association Graph

```
//Define Original Policy

//"medtronic device may communicate with nexus5x devices"
pred Rule1allowance[src, dst: Device]{
        ((src.maker = Medtronic) && (dst.model = Nexus5x)) ||
        ((src.model = Nexus5x) && (dst.maker = Medtronic))
}
```

Figure 9.4: Original Policy in Alloy

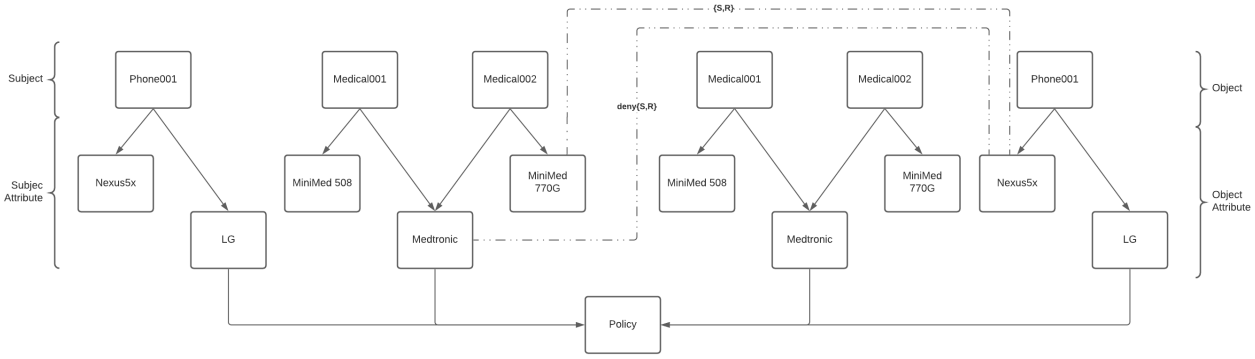

Figure 9.5: Altered Policy Assignment & Association Graph

```
//Define Altered Policy

//"medtronic device may not communicate with nexus5x devices"
pred newRule1denial[src, dst: Device]{
        ((src.maker = Medtronic) && (dst.model = Nexus5x)) ||
        ((src.model = Nexus5x) && (dst.maker = Medtronic))
}


pred Rule2allowance[src, dst: Device]{
        ((src.model = MiniMed770G) &&(dst.model = Nexus5x)) ||
        ((dst.model = MiniMed770G) &&(src.model = Nexus5x))
}
```

Figure 9.6: Altered Policy in Alloy

```
//Check for conflicts

assert noConflictAfterAlteration{
        some src, dst: Device | Rule2allowance[src,dst] && not (newRule1denial[src,dst])
}

check noConflictAfterAlteration for 5
```

Figure 9.7: Conflict Check Assertion on Altered Policy

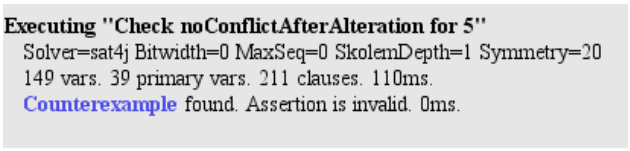

Figure 9.8: Scenario Conflict Identification Result

Chapter 10

Future Work

10.1 Process Automation

Currently most steps fn our process are manual. This includes deriving environment entities, relationships, policy properties, translating NGAC to Alloy, generating an Alloy environment. In the future it would benefit from automating such tasks. Doing this would assist in creating an all-in-one solution which only takes input from the user and has little human interference.

It may be easiest to automate the NGAC translation due to the fact that we have already defined an algorithm for this action.

Most difficult step to automate may be rule set generation. This step is highly dependant on the users security goals and a translation form common language to a form which can be easily processed.

10.2 Stateful modeling

Our current work deals with static snapshots of IoT environments and policies. In order to handle access control which involves past or future states, we need to adapt our policy model to include these state transitions. There are a number of requirements that need to be met in order to model stateful access control. First, the state transitions need to be defined which also includes defining the scope of the environment. Our work occurred at a high level network perspective, changing this scope will influence the environment and policy definition stages and trickle down to the rest of the process.

Through our work we have defined a number of stateful rules that may be used in a smart home setting. An example of a stateful rule would be "if Light001 was received information from Phone001 then Light001 cannot send to Phone002". This rule takes the form of if X then Y where X is a statement about a past event and Y is a traditional access control statement like the ones seen

in this work. Similar to the rule archetypes defined in this work it is possible to define an archetype for stateful rules as well.

10.2.1 Dynamic Conflicts

Dynamic rule conflicts may occur when a policy is already defined, and the environment changes to create a conflict. An example of this dynamic conflict would be a situation where a smart home policy is already defined such that access is provided or restricted based on location or time. This type of conflict has been demonstrated and explored in [10]. As the time of day changes, or the location of the device moves, there may be conflicts with defined policy rules. In conjunction with stateful modeling, there is an opportunity to explore detection or prediction of dynamic conflicts.

10.3 Alternative Model Checkers

There exist alternative model checkers referenced in our related work section which need to be further explored in the context of IoT environments. That being said, these alternatives have proven to be either underdeveloped for this use case, or out of support for modern usage. Another direction this work could take would be leveraging other existing access control tools into this process such as ACPT by NIST referenced in our related works section.

10.4 Fine Grain Access

Currently our solution predicates on simple network control such as send and receive messages. However, our models can be modified to include various degrees of granularity whether it be specific messages, details within those messages, or collections of messages.

This work provides potential for exploration of fine grained access control modeling. Each device in our smart home model sends and/or receives controlling messages. This is how the user orchestrates and, growing more popular everyday, automates the smart home. Because we are modeling the communication between devices, information inherent to this communication is at

our disposal when modeling the environment and in turn the access controls. Here in Fig. 10.1 we define an abstract representation of the messages corresponding to each device. These messages have been observed through the operation of each device. Each message modeled represents an interaction where a collection of packets were send and received. With these modeled messages we gain the ability to incorporate them into an access control policy and furthermore our conflict and redundancy identification methods.

In order to create the message model in Alloy we required an enumeration of all possible controls that these devices could send and receive. The device messages in Figure 10.1 were enumerated from an IoT testbed environment in the Colorado State University IoT Security Lab. We performed each functionality of the device as intended by the manufacturer one by one and recorded the traffic through an application called t-shark. Once the network traffic for each device was collected into .pcap files and organized, we manually curated the communication to the unique sets of packets which made up each function. Finally we organized each set into the aforementioned messages seen in Figure 10.1.

```
// Define Device Messages
abstract sig MSG {}

//Define messages expected by TPLink light devices
one sig DEFAULT_PAGE_LOAD, DEVICE_INFO_LOAD, DEVICE_SELECT,
ENABLE_REMOTE_CONTROL, ENERGY_PAGE_LOAD, FIRMWARE_UPDATE,
ONPS_PAGE_LOAD, PRESET_DIM, PRESET_PAGE,
SHCEDULE_ADD, SCHEDULE_PAGE, SETTINGS_PAGE_LOAD,
COLOR_SLIDER, LIGHT_ON, LIGHT_OFF, ACK, CHANGE_COLOR, DIM, BRIGHT extends MSG{}

//Define messages expected by Arlo camera devices
one sig RECORD, SNAPSHOT, ARM, DISARM, NEW_MODE,NEW_MODE_SELECT,
SCHEDULE extends MSG {}

//Define messages expected by idevice dimmer devices
one sig DIM_ADD_SCHEDULE, DIM_LIGHT_DEC, DIM_LIGHT_INC, DIM_NIGHT_INC,
DIM_NIGHT_DEC, DIM_NIGHT_OFF, DIM_NIGHT_ON, DIM_LIGHT_ON, DIM_LIGHT_OFF,
DIM_PRESET_COLOR, DIM_SATURATION_INC, DIM_SATURATION_DEC, DIM_SLIDE_COLOR,
DIM_UPDATE extends MSG{}

//Define messages expected by Samsung TV devices
one sig TV_ON, TV_OFF extends MSG{}
```

Figure 10.1: Alloy Device Control Messages

Chapter 11

Conclusion

In this work we have defined a process for policy analysis through conflict/redundancy identification. We also provide a technique for identifying partial conflicts. We demonstrated the applicability of this process to the Next Generation Access Control framework with the use of Alloy specification language and Alloy Analyzer model checking software. This work has illustrated the procedure taken to translate access control policies to NGAC and furthermore the novel translation of NGAC to Alloy specification language. This work has defined Alloy assertions that can be adapted to fit any modeled IoT environment given our modeling conventions. Finally, this work demonstrated the modeling and analysis process through a practical scenario involving the preservation of access to medical IoT devices within the home.

This work has provided a foundation for expansion in policy modeling techniques such as stateful policies, user interaction to enable policy analysis, and automation for an effective policy checking tool.

Bibliography

[1] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, August 2017. USENIX Association.

[2] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 542–553, New York, NY, USA, 2015. Association for Computing Machinery.

[3] Rob van der Meulen. Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016, Feb 2017.

[4] Mark Hung. Leading the iot. https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf.

[5] Daniel Jackson. Alloy: a language and tool for exploring software designs. *Communications of the ACM*, 62:66–76, 08 2019.

[6] David Ferraiolo, Ramaswamy Chandramouli, Rick Kuhn, and Vincent Hu. Extensible access control markup language (xacml) and next generation access control (ngac). In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control - ABAC '16*, 2016.

[7] David F. Ferraiolo, Serban I. Gavrila, Wayne Jansen, and Paul E. Stutzman. Policy machine: Features, architecture, and specification. 2015.

[8] Soufiane Zahid, Abdeslam En-Nouaary, and S. Bah. Practical model checking of a home area network system: Case study. *Journal of Computing and Information Technology*, 27:1–16, 2019.

[9] Phillipa Bennett, Indrakshi Ray, and Robert France. Modeling of online social network policies using an attribute-based access control framework. *Information Systems Security Lecture Notes in Computer Science*, page 79–97, 2015.

[10] Manachai Toahchoodee and Indrakshi Ray. On the formal analysis of a spatio-temporal role-based access control model. *Lecture Notes in Computer Science Data and Applications Security XXII*, page 17–32, 2008.

[11] Ang Li, Qinghua Li, Vincent C. Hu, and Jia Di. Evaluating the capability and performance of access control policy verification tools. In *2015 IEEE Military Communications Conference*, 2015.

[12] Information Technology Laboratory Computer Security Division. Acpt - access control policy testing.

[13] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, page 196–205, New York, NY, USA, 2005. Association for Computing Machinery.

[14] Karthick Jayaraman, Vijay Ganesh, Mahesh Tripunitara, Martin Rinard, and Steve Chapin. Automatic error finding in access-control policies. In *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[15] Evan Martin, Jeehyun Hwang, Tao Xie, and Vincent Hu. Assessing quality of policy properties in verification of access control policies. In *2008 Annual Computer Security Applications Conference (ACSAC)*, 2008.

[16] Marinos Charalambides, Paris Flegkas, George Pavlou, Javier Rubio-Loyola, Arosha Bandara, Emil Lupu, Alessandra Russo, Naranker Dulay, and Morris Sloman. Policy conflict analysis for diffserv quality of service management. *IEEE Transactions on Network and Service Management*, 6(1):15–30, 2009.

[17] E.c. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[18] Jonathan D. Moffett and Morris S. Sloman. Policy conflict analysis in distributed system management. *Journal of Organizational Computing*, 4(1):1–22, 1994.

[19] P. Zave. A practical comparison of alloy and spin. *Formal Aspects of Computing*, 27(2):239–253, 2014.

[20] Fortinac. https://docs.fortinet.com/document/fortinac/8.6.0/administration-guide/659283/fortinac.

[21] Ieee standard for local and metropolitan area networks–port-based network access control. *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)*, pages 1–289, 2020.

[22] Vijay Sivaraman, Hassan Habibi Gharakheili, Arun Vishwanath, Roksana Boreli, and Olivier Mehani. Network-level security and privacy control for smart-home iot devices. In *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications*, 2015.

[23] Ravi S. Sandhu. Role-based access control. *Advances in Computers*, page 237–286, 1998.

[24] Yu Chen, Ronghua Xu, Erik Blasch, and Genshe Chen. A federated capability-based access control mechanism for internet of things (iots). *Sensors and Systems for Space Applications XI*, Feb 2018.

[25] Bruhadeshwar Bezawada, Kyle Haefner, and Indrakshi Ray. Securing home iot environments with attribute-based access control. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, 2018.

[26] Sanghak Lee, Jiwon Choi, Jihun Kim, Beumjin Cho, Sangho Lee, Hanjun Kim, and Jong Kim. Fact. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, 2017.

Appendix A

Asynchronous Model

Our work disregards modeling device communication within IoT networks. However, we have done some rudimentary work with modeling the communication of IoT devices as an initial step in SPIN verification.

When modeling a networked device we need to define the 1) input language 2) set of input channels 3) set of input tasks, set of state variables. Note that for this simple model we do not need to define output channels and their corresponding output tasks as this device only takes some input and changes the physical state as a response.

In order to represent the networked behavior of the TPLink light bulb we define a language of type MSG. The alphabet of MSG is listed in table A.1 and represents one "letter" for each unique function control that the device can accept. Only these words are recognised by the device and any other inputs are ignored. This device also has only one input channel. This channel represents the network interface card that data is being received from. Finally the light module has a set of state variables containing one variable that will be used to store the previous message input to the device.

Formally we can define the device as:

Table A.1: MSG Language for TPLink Light

MSG word	**Function Description**
CallHome	Device sends and receives data from manufacturer server
OnFromAppPage	Loading the "on from app" pages in the settings section of the controlling application
DeviceInfoPage	Loading the "device info" page in the settings section of the controlling application
DeviceSelect	Selecting the LB100 device to access control options
DNSQuery	Device autonomous dns query for ntp and manufacturer servers
EnableRemoteControl	Toggle "enable remote control" option in the settings page of the controlling application
EnergyPage	Loading the energy usage page in the controlling application
IdleCallback	Device autonomous action receiving and sending data from the controlling application. This is initiated by the controlling application.
IdlePing	Device sends a single UDP packet to the controlling application. This is initiated by the device
NTPQuery	Device sends an NTP request to an NTP server
OFF	Device ordered to turn off the light
ON	Device ordered to turn on the light
OnFromPowerSourcePage	Loading the "on from power source" page in the settings section of the controlling app
PresetDim	Selecting a predefined light dim value
PresetPage	Loading the dim presets page in the controlling application
ScheduleAdd	Adding a scheduled time for the light to alter its state
SettingsPage	Loading the settings page in the controlling application
SliderTap	Setting the dim value through the controlling application's slider mechanism

TPLINK$(I, S, Init, A)$

$I : \{in\}$

$O : \{out\}$

$S : \{Hold\}$

$Init : (Hold = null)$

A : set containing one task for each unique letter in the alphabet

(A.1)

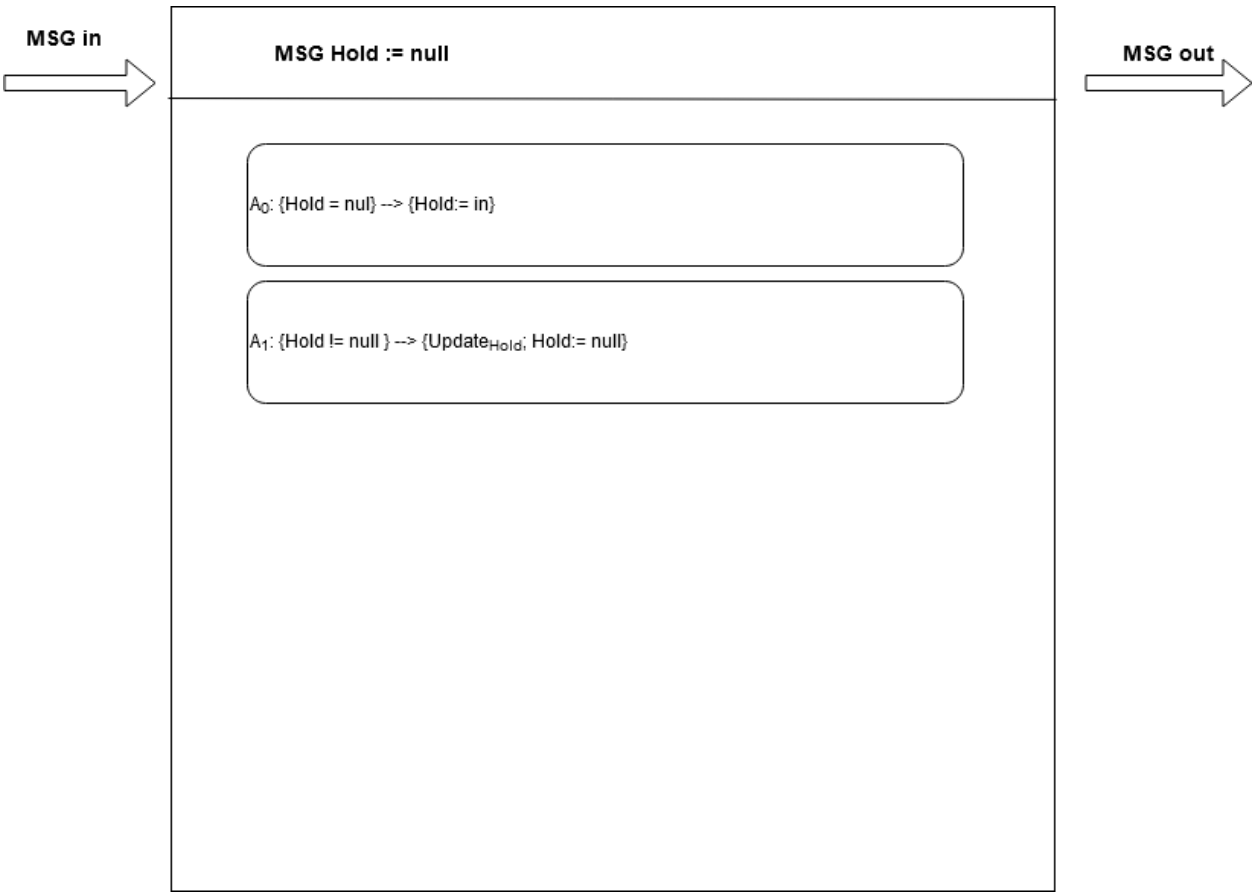

Figure A.1: Asyncronous Model For IoT Device

We graphically represent the TPLink Light as shown in Figure A.1. In this depiction there are two tasks that cycle between each other. The first task is designed to take in an input at set the state variable to that MSG. This task is necessary to be able to ensure that each letter is processed. The second task performs an update function based on the value of the state variable. After this update function runs the state variable is reset back to *null* and an *ACK* message is sent out.

However, this is only a simple module that describes the basic behavior of the device. In reality this module ignores many nuances of the device in its entirety.