Designing Networked Objects to Achieve Reasonable Security

by

Zane Alexander Markel

B.S., Computer Science
United States Naval Academy, 2015

Submitted to the Institute for Data, Systems, and Society
in partial fulfillment of the requirements for the degree of

Master of Science in Technology and Policy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author . . . .
Institute for Data, Systems, and Society
May 12, 2017

Certified by . . . .
David D. Clark
Senior Research Scientist, CSAIL
Thesis Supervisor

Accepted by . . . .
W. A. Coolidge Professor Munther Dahleh
Director, Institute for Data, Systems, and Society

Designing Networked Objects to Achieve Reasonable Security

by

Zane Alexander Markel

Submitted to the Institute for Data, Systems, and Society
on May 12, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Technology and Policy

Abstract

To maximize the value of the Internet of Things (IoT), developers need to build devices that balance security with features, cost, and usability, relative to the threats that their particular devices will face. However, many IoT devices have thus far failed to achieve this balance. Various organizations have published copious security frameworks to help developers. Of these, frameworks that focus on desirable outcome metrics remain theoretically desirable yet infeasible to use in practice. The other frameworks, which focus on some aspect of the development process itself, are widely used despite a lack of methods for determining their utility.

This work introduces six criteria useful for evaluating and comparing these process-based frameworks. Applying them to multiple security frameworks, we find that these frameworks often derive from inflexible conceptions of security, limiting the ability of developers to to vary their security designs. Even when developers are given options, they lack the tools necessary to balance security with other tradeoffs respective to the situations their products will be used in. To begin to address these shortcomings, we propose the Processes for Reasonably Secure Design (PRSD), a novel process-based security framework that helps developers comprehensively and systematically consider the security threats an IoT device may introduce to its surroundings, options for mitigating those threats, and the tradeoffs between those options. To demonstrate its worth, we apply it in multiple case studies. Further, using the six criteria, we evaluate PRSD and find that, in addition to providing useful and novel guidance, it has practical qualities that could make it suitable for many real development efforts.

Thesis Supervisor: David D. Clark
Title: Senior Research Scientist, CSAIL

Acknowledgments

I would like express my deep gratitude to Dr. David Clark for inspiring my research direction, keeping me critical and curious with our formative conversations, and keeping me on course throughout these last two years.

Additionally, I would like to thank the many people in the MIT Internet Policy Research Initiative who I've worked with—without your ideas and support, this thesis never would have been written.

Further thanks go towards the faculty in the Technology and Policy Program, whose lessons profoundly shaped the perspective that fueled this work, and to all the staff, who keep everything together.

Contents

List of Figures

Chapter 1

Introduction

1.1 Internet of What?

Developers everywhere are producing objects embedded with network connectivity. This *Internet of Things* (IoT), which includes network-connected[1] objects that are not traditionally conceptualized as computers, could lead to devices with the ability to generate, transfer, and act upon data with minimal human interaction throughout all facets of life [45]. These abilities could enable widespread automation and coordination, from self-driving cars that coordinate to get through traffic efficiently to home devices that take care of household chores while the homeowners are away. Expectations are high: Currently, over 4.5 million people are working as developers in this space [13], and projections estimate that the world could house up to 100 billion Internet connected devices by 2025 [44], which, by one projection, could have an economic impact of \$3.9 to \$11.1 trillion per year [58].

Although experts have made a wide variety of predictions about how the Internet of Things will yield that added value, there is widespread consensus that security should be of the utmost concern throughout the development of these networked devices. Groups ranging from the Internet Society [45], to the Department of Homeland Security [88], to a diverse panel of business, academic, and legal experts put together

[1]Despite the name, not all IoT devices must be part of the public, globally accessible Internet. IoT could comprise multiple partitioned networks.

by the FTC [26] have all publicly raised grave security concerns about IoT devices. Weak security could lead to serious widespread harm. For instance, hacked coordinating automated cars could cause fatal collisions, and seemingly innocuous chore-doing devices could be leveraged to get into a connected door lock, leaving people's homes open to burglary. If security is bad enough, it could lead to a collapse in consumer trust of IoT, which could ruin businesses and deny many people an increased quality of life.

The IoT devices that have been produced thus far have not done much to dispel these concerns. In fall 2016, malware named *Mirai* spread to at least 493,000 devices spanning at least 164 countries [92, 57]. Together, these infected devices launched several of the most disruptive distributed denial-of-service (DDoS) attacks in history. The attacks even temporarily shut down Dyn, a major DNS provider for the United States' east coast [90]. Later analysis discovered that most of the "bots", or infected machines, compromised by Mirai were home IoT devices like web cameras and DVRs [92]. Thus, the IoT has already made traditional computer security attacks more powerful.

Additionally, researchers have shown that existing IoT devices have enabled attacks that are not even possible through traditional computers. For instance, in 2015, security researchers Charlie Miller and Chris Valasek showed that they could scan for, identify, and exploit Jeep Cherokees, even while they are driving at full speed. Through this exploit, the researchers were able to take control of critical systems like braking and steering [59]. Likewise, Runa Sandvik and Michael Auger demonstrated at BlackHat 2015 how they found a way to remotely take control of an automatically-aiming IoT rifle in order to simply brick[2] it or to change the target that the rifle aims at [73]. As it stands, there is little reason to think that future IoT products will buck this trend of insecurity.

[2]A "bricked" device has been attacked so badly that it has been rendered more or less permanently unusable.

1.2 Purpose

The IoT space is advancing rapidly, and there is not much time before standards lock in and norms for design patterns take shape. In order to ensure that these standards and norms facilitate secure outcomes, it is imperative to explore the IoT security space now. This thesis contributes to that search by addressing the following question: Is there a procedure that developers can follow to design the Internet of Things with reasonable security?

This thesis answers that question in the affirmative by proposing the Processes for Reasonably Secure Design (PRSD), a framework that enables developers of home IoT products implementing the communication specifications of the Open Connectivity Foundation to systematically identify and weigh security considerations and their tradeoffs in an effective, flexible, cost effective, and scalable way.

In the process of unpacking that question and supporting PRSD as an answer, this thesis makes the following contributions:

1. It explores the concept of "reasonable security" and derives criteria that a procedure must meet in order to help developers best achieve it through their work.

2. It demonstrates systematic errors in current security guidance available to home IoT developers, including both standalone guidance and guidance derived from standardized architectures.

3. It proposes PRSD, a novel framework which developers can rely on to mitigate some of the systematic shortcomings in security guidance they face and can adapt to changes in the IoT space over time.

The remainder of this thesis is organized as follows. Chapter 2 explores the background material, establishing a definition of "reasonable security", going into more detail about the current state of the Internet of Things, and outlining some of the existing security design guidance. Next, Chapter 3 derives criteria for evaluating security design guidance and applies them in a high-level evaluation of the guidance referenced in the previous chapter. After that, Chapter 4 introduces PRSD, summarizing the framework and its application in a case study. The complete framework and case studies are presented in full in the appendices. Following that, Chapter 5 evaluates PRSD using the same criteria previously applied to existing security design guidance. To do so, the chapter uses conclusions drawn from the case studies and analyses economic and legal factors of using PRSD. Finally, Chapter 6 draws final conclusions and notes promising potential future work.

Chapter 2

Background

2.1 What is Security?

In the Internet of Things, as with most things, perfect security is neither feasible nor desirable. People take risks every day, because either the risks are unavoidable or they believe that the potential benefits outweigh the risks. Internet of Things security is no different—the only way to achieve perfect IoT security is to avoid the IoT altogether. Ideally, stakeholders will want maximum utility from an IoT system. Security helps to deliver this utility by mitigating the risks faced in the operation of the system. However, it's possible that security measures, while mitigating risks, will inadvertently limit the benefits of an IoT system. These limitations occur when the security measures constrain the systems' functional capabilities, costs stakeholders time or resources, or makes the system more cumbersome to use. Optimal security, then, should mitigate risk just to the point of maximizing overall system value. We term this optimum *reasonable security*.

More specifically, given an IoT device or network, a stakeholder[1] of that device or network, assets of the stakeholder affected by the device or network, and potential risks to those assets stemming from the device or network,[2] the device or network is *reasonably secure* when those risks are minimized so as to maximize the overall

[1]A stakeholder is any person or group with an interest in or is affected by something [29]. In IoT, this includes, among others, owners, users, producers, and even attackers.

[2]Security particularly focuses on risks that result from adversarial intent.

expected value of the device or network to the stakeholder. From this definition, we can immediately see that reasonable security varies with the perspective of the particular stakeholder, that stakeholder's assets, and the environment that the IoT system and those assets operate within. Since the IoT, like networks in general, inherently involves interactions between different agents, misalignments in their interests can lead to different or even conflicting states of reasonable security.[3] Furthermore, more valuable assets and a more adversarial environment justify more expensive or onerous security measures. Thus, reasonable security is only a coherent construct when it is gauged with respect to these three varying factors.

Any attempt to distill a single "objective" metric or standard of security is actually just a single non-universal perspective of security, which may be reasonable or unreasonable. Nevertheless, many sources traditionally define security this way. For instance, ISO standards define security in terms of rigid adherence to the standard pillars of confidentiality, integrity, availability [49]. While these three are often laudable security goals, treating them as isolated and absolute goals ignores the tradeoffs between them, other security goals (such as accountability), non-security design goals (such as cost), and the perspectives and assets of other stakeholders. Likewise, a variety of the security literature puts the onus on design specifications, more or less stating that a system is secure if and only if it adheres to specification for all inputs (for example, see [39, 63].) From this perspective, developers are told to specify, or formally articulate, the intended proper behavior of their proposed product for all possible inputs; if the realized product adheres to these specifications, it is secure. This framing of security is not helpful, as it presumes that there is a single, known specification that achieves reasonable security for all stakeholders, assets, and environments; however, stakeholders' needs vary, and there often exist security risks that developers never thought of during specification. More generally, these "objective" security concepts can be useful as heuristics that help to estimate and strive for reasonable security, but they should not be conflated with it.

Even at times where there is public or expert consensus that one feature or an-

[3]Clark and Blumenthal make this argument for the Internet in general [18].

other is *always* good and warranted, it may be that there exists various stakeholder assets and contexts that demand differing security measures. For example, imagine an Internet-connected "smart plug" that serves as an intermediary between a power source and a device that relies on power from that source to operate. This smart plug would allow or disallow power through the source to the device according to commands sent over the network. Consider what security measures would be appropriate to protect the smart plug when its used to power a lamp in a person's home versus a nuclear Intercontinental Ballistic Missile launch tube. In the former situation, even having to type in a password to turn the light on and off may impose an unreasonable burden on the homeowner; in the latter, if the smart plug could be justified at all, a military would likely demand stringent protections that require input from multiple authorities. Between these situations, frequently-given security advice ("always use random, strong, and unique passwords" [30]) can be misleading for seemingly innocuous IoT devices.[4]

More generally, there is a body of literature that argues that common expert security advice is often inappropriate. For instance, Cormac Herley demonstrated that adhering to common password strength rules, reading URLs to detect phishing, and simply heeding certificate error warnings are almost always economically irrational [41]. It follows from this that devices that put these responsibilities on users are not reasonably secure. Moreover, another study points out that the common password advice to only use random, strong, and unique passwords is actually infeasible for people to follow properly, so rationality cannot even be considered [30]. Anne Adams and Angela Sasse showed that users often deal with impracticable advice and policies by finding ways to circumvent them [3]. Generalizing further, Ross Anderson has made a career out of arguing that perverse incentives are at least as much at the root of information insecurity as technical shortcomings, demonstrating his case by pointing out common microeconomic problems that have plagued information security, such as moral hazard, asymmetric information, and adverse selection [6]. To

[4]It also happens that there are actual smart plugs that one can buy today, and at least one model is incredibly insecure [34].

summarize, this literature argues that, when designing for security, developers need to weigh the anticipated benefits and costs of their options, choosing the options that result in the most reasonable security for the stakeholders they care about. Advice from security experts that is divorced of any consideration of stakeholders, assets, and context is therefore not useful for thinking rationally about reasonable security. Instead, developers need tools that help guide them through the process of weighing the benefits and costs of security options.

2.2 The State of the Internet of Things

2.2.1 Security by Design

A key aspect of weighing security benefits and costs is *security design*, which comprises the *security policy* and selection of *security mechanisms* intended to enforce the security policy. The security policy for an IoT system describes what agents, which includes stakeholders (benign and malicious) and their devices, are allowed and not allowed to do in the system, including the actions they can take and the objects those actions can target.[5] In other words, it is the plan for acceptable behavior that developers try to achieve in order to manage risk to stakeholder assets. Security mechanisms, then, include any processes or machinery, technical or otherwise, put in place to enforce the security policy. To be clear, while security itself is relative to a particular stakeholder, assets, and environment, the security policy and the security mechanisms of an IoT device are intrinsic properties of that device. Whether explicitly formalized or implicitly understood, the developers and administrators' intents when they design and configure a device reflect its security policy. Likewise, the actual mechanisms that result from that design and configuration are the device's security mechanisms. The challenge of security design, then, is to choose a security policy and security mechanisms that make the security of a product or system as

[5]Both benign and malicious actors must be included in the security policy, because both types of actors need permissions and restrictions.

reasonable as possible across all valued stakeholders, assets, and environments.[6]

Currently, security design is particularly challenging for the Internet of Things. Since it combines physical sensing and actuation with all the elements of traditional computing and networking, the IoT is inherently more complex than the traditional Internet. As a result, developers have a harder time anticipating how individual actions will impact a system, so it is tougher to devise a smart security policy or to choose appropriate security mechanisms. Additionally, the highly uncertain future of the IoT makes security design even more difficult; when designing a product, it is difficult to anticipate what other devices and services the product might eventually interact with, so it is difficult to estimate threats appropriately (neither wasting resources to protect against threats that would never materialize nor failing to protect against those that do.) Compounding this problem further, many current IoT developers may be coming from industries where they never had to consider information security before, leaving them ill-equipped to make these decisions smartly.

Poor security design enabled some of the biggest IoT attacks that have occurred. Mirai, the malware responsible for some of the most disruptive DDoS attacks in history, operated by taking advantage of common security design mistakes. Namely, the devices it infected ran SSH or Telnet—services that enable remote login and operation of the underlying operating system—and used known default usernames and passwords for the administrator accounts. Mirai takes advantage of this by scanning the Internet for devices listening with SSH or Telnet, trying to log in with a small list of known default usernames and passwords, and then installing itself onto any machine it gets on to [92]. There are no software bugs involved in this process; Mirai takes advantage of a flawed security design. The developers of vulnerable devices do not consider how an attacker might find their default passwords and take advantage of listening SSH and Telnet services in order to harm stakeholder assets. In the case of Mirai, the assets harmed are the targets of DDoS attacks, and the stakeholders to those assets are the users and owners of those targets. However, such

[6]It is up to developers to decide the relative weights of various stakeholders and assets. Such determinations are outside the scope of this work.

an attack could easily, among other things, steal sensitive data or simply brick the device (in either case, harming the asset of the device's owners and users.) Even if the device developers only care about their own assets, the bad reputation—and resulting damage to the bottom line—is surely something they would want to avoid. All it would take to stop Mirai would be to not run SSH and Telnet, which are not necessary to these devices' operations, anyway. Turning off these services would have significantly mitigated risk to the public and the producer's reputations with negligible downsides. Consequently, the devices would have more reasonable security from the perspective of the public and the producers.

The security design of the original TrackingPoint Rifle was also problematic. Researchers found that the rifle has an unpublished administrative application programming interface (API) with functions that, among other things, grant full access to the operating system over SSH (password required) and allow an external source to modify parameters like wind, temperature, and ballistics information that the rifle uses to aim at a target. Any device on the same network as the rifle can send it commands through this API, and, aside from the SSH login command, the rifle will accept them without any verification or authentication. Separately, the rifle accepts software updates that are signed with one of two GPG keys. The security researchers found that one of these keys is unchanging and hard-coded locally on every TrackingPoint rifle, and they were able to lift it from one that they owned. With it, one can send any arbitrary software update to a TrackingPoint Rifle. With knowledge of either the unpublished administrative API or the GPG key, along with network access to a TrackingPoint Rifle, the researchers demonstrated that an attacker can make the rifle miss its target (or hit another one), incorporate it into a botnet, or simply brick it [73]. These vulnerabilities put lives at risk. TrackingPoint's business depends heavily on a reputation for reliability, so they would have a strong interest in preventing these critical vulnerabilities even if they only care about their own assets. Furthermore, there are security mechanisms that could largely mitigate these risks with little sacrifice in cost, functionality, or usability. Thus, from the perspective of either the users or the TrackingPoint, the security design of these rifles clearly fell short of reasonable security. While it is not clear exactly what led the developers to make these design mistakes, a better procedure of systematically identifying security threats (including their likelihood and the resources that attackers might expend to pull off attacks), potential mitigating security mechanisms, and balancing the mechanisms against tradeoffs would have greatly diminished the likelihood that the developers would have made these mistakes.

The 2014 Jeep Cherokee also resulted from poor security design. Modern Jeep Cherokees are somewhat complex IoT systems in themselves; they connect to the Sprint cellular network and have many interoperating sub-components. Communication between processes running within the car, from the media player to braking, is handled by a process called D-Bus. The sub-components of the car fully trust D-Bus commands, obeying them without any sort of verification. This may have been a reasonable assumption before Jeep started connecting their cars to the Sprint network, but it no longer holds: D-Bus has an IP address accessible to the entire Sprint network, and anything on that network can connect to it over port 6667. Further, D-Bus itself trusts any external communications without authentication. The researchers were able to exploit this misplaced trust to gain control over many critical systems, like braking and steering [59]. Lone individuals and hacktivist organizations would not have a hard time causing real damage by carrying out this exploit. Although the likelihood of such an attack is somewhat low (relative to DDoS attacks, spam, and identity theft operations), Jeep could have put security mechanisms in place to prevent this exploit for virtually no cost or sacrifice in functionality or usability. Indeed, Jeep eventually patched this particular vulnerability simply by blocking traffic over port 6667 on the Sprint network [59]. It's difficult to imagine that the Cherokee's designers intentionally put this security design in place; they probably either falsely assumed that arbitrary connections over the Sprint network were impossible or misunderstood the chain of trust the 2014 Jeep Cherokee's components implicitly relied upon. If the teams designing the vehicle had explicitly laid out the capabilities of the systems they developed and merely brainstormed about how an attacker could take advantage of those capabilities through the systems' interfaces, the vulnerabilities would have been obvious. However, it takes careful organization of a system's designs to see these things. As it is, this case demonstrates how uncertainty and complexity in IoT systems makes it particularly difficult to gauge the impact of design decisions on overall security.

Certainly, there are other aspects to security besides design, but they are not the primary focus of this thesis. Traditionally, many security vulnerabilities arise from implementation bugs. These vulnerabilities arise from a discrepancy between developer intentions and the resulting code that attackers can exploit. (For contrast, vulnerabilities in security design occur when an attacker can harm stakeholder assets while operating within the bounds of the developer intentions [40].[7]) While implementation bugs will certainly be a problem, it's not yet clear if or how IoT-specific implementation bug vulnerabilities differ from traditional implementation bug vulnerabilities. Additionally, human factor weaknesses are also a major security weakness in many computing systems. Human factor attacks rely on social engineering, such as phishing attacks, to exploit a system (see, for example, [7].) Human factors are related to security design in that smart security design will account for actual human behavior, although there are aspects of human factor attacks that fall outside the scope of an IoT device's security design. Regardless, since IoT is characterized in large part by automated interactions between machines with minimal human involvement, human factors may be less problematic for IoT security than for traditional computing. While implementation bugs and human factors may actually turn into important research subfields of IoT security, there is less reason to give them as much priority as security design at this time.

2.2.2 Current Trends in Internet of Things Designs

The Internet of Things is new and immature, and it is uncertain what security designs will come to dominate. Perhaps unsurprisingly, the millions of people working on IoT

[7]Isolation, which involves separating components of a system to minimize the ability of vulnerabilities in one to propagate through another, spans both design and implementation. There are probably many unique isolation challenges that IoT developers will face. The remainder of this thesis will consider isolation as needed.

development are employed by a wide variety of companies and organizations, each with their own vision about how the Internet of Things should operate and how to secure it. The home IoT, by itself, is no exception. Headline-grabbing products like the Nest thermostat, a smart thermostat produced by the eponymous startup [62], and products marketed by tech giants like the Echo, a smart speaker produced by Amazon [5], capture most of the IoT analysis and zeitgeist. However, the tens of thousands of products that contributed to the Mirai botnet demonstrate that many of the IoT products actually leaving the shelves are from the long tail of lesser known IoT devices.

Devices currently in production generally all follow their own proprietary procedures for communicating with other networked entities. While they may use standard low-level communications technology (such as ZigBee [93], CoAP [19], Bluetooth, and WiFi), their application programming interfaces are not always designed to be interoperable. Indeed, many IoT devices can only be operated by a smartphone app designed to operate just that device. For instance, Fitbit [28] and Garmin [33], Sonos [81] and Ultimate Ears [86], and OnStar [64] and Tesla [84] each have their own mutually incompatible systems for communicating with IoT watches, speakers, and cars, respectively. As it stands, the Internet of Things is not as inter-networked as the name suggests. This isolation is socially suboptimal, as many of the potential benefits of the Internet of Things require interoperability. Moreover, for the purposes of this work, it makes it difficult to reason about security designs beyond individual point cases.

There are a wide variety of efforts aimed to make devices from different brands interoperable, each with their own security implications. Amazon has garnered the most attention lately with Alexa, a voice-operated cloud service that acts like a personal assistant [5]. While initially limited to powering the Amazon Echo, Amazon has allowed other developers to create "Skills" that allow Alexa to interact with other cloud services and IoT devices. For instance, an IoT washing machine developer could design Alexa Skills so that a homeowner could operate her washing machine through voice commands to an Amazon Echo. Additionally, Amazon now allows other devices to run Alexa themselves. Many companies have taken advantage of this opportunity. The 2017 Consumer Electronics Show featured IoT prototypes across many brands and functions that either ran Alexa or were designed to have Alexa Skills [50]. It's unclear how secure an Alexa-controlled world would be. Amazon does not advertise security policies heavily, and it could easily change security features at its own whims. Indeed, Amazon quietly removed encryption from their products in 2016 [31].

Google and Apple are each developing their own answers to Amazon's services. Recently, Google released the Google Home, an smart speaker similar to the Amazon Echo [36]. The Google Home runs the Google Assistant, an AI personal assistant in the vein of Alexa, which supports third party "Actions" that are much like Alexa Skills [35]. Google is more clear about security than Amazon: they claim Google Home data is "protected by one of the world's most advanced security infrastructures"[8] and uses "encryption by default" [24]. Unlike Amazon and Google, Apple is taking a less AI-focused approach to IoT with Homekit, a service that provides a common interface between household IoT devices and iOS devices like iPhones and Apple TVs [9]. Apple is offers some details about Homekit security. Homeowners are responsible for determining what objects to trust, and all users of a given Homekit network are given full access to all services on all connected Homekit devices. Communication between devices is end-to-end encrypted [10, 8].

Other efforts are bringing together a wide variety of stakeholders in order to bolster interoperability between IoT devices. The Internet Engineering Task Force (IETF), the standard-setting organization responsible for many fundamental Internet standards like TCP/IP [79], has held multiple workshops about IoT interoperability and security [11, 90], and are in the process of developing IoT standards. However, nothing is finalized.

In a separate initiative, the Open Connectivity Foundation is a coalition of private firms pursuing a common communications architecture to allow interoperability between all compliant devices. There are over 300 member organizations (and counting), including Cisco, Intel, and Microsoft, and they have released a draft of their

[8]For more information on the security design of Google's infrastructure, see [37].

communications standard [69, 67]. Furthermore, a few devices have been certified to be compatible with these standards [65]. Since the OCF standard is currently the open IoT communications architecture with the greatest industry support, we will examine the security implications of this architecture in more detail throughout the remainder of the thesis.

The Open Connectivity Foundation Communications Architecture

The Open Connectivity Foundation communications architecture comprises a middleware communication protocol and a data model to facilitate communications between IoT devices, applications, and services. The communication protocol is technology-neutral, abstracting away the particulars of the hardware, operating systems, and networking protocols (which could encompass multiple layers, including TCP/IP, Wi-Fi, ZigBee, CoAP, and others.) The idea is that any two OCF *endpoints*, or OCF-compliant entities, on the same network should be able to seamlessly communicate with each other without any special configuration to each other, regardless of differences in hardware, operating system, or application. This is comparable to how, through the World Wide Web, any compatible device can easily communicate with any web server.

To accomplish this, the OCF specification has endpoints represent their exposed physical elements and data as "resources." Endpoints can access resources through a RESTful request-response communication model.[a] Anything making a request for a resource is termed an "OCF client", and anything that responds to a request—usually the endpoint that controls the requested resource—is termed an "OCF server." If necessary, "OCF intermediaries" act as routers to deliver traffic between OCF clients and servers that are not directly connected. Any endpoint can be an OCF client, server, or intermediary at any time, depending on their role in sending, routing, or responding to requests. Since all operations are RESTful, each interaction contains all the necessary context; in other words, no state is maintained. Furthermore, all operations derive from five generic operations: Create, Read, Update, Delete, and Notify. For instance, an OCF air conditioner might want to check an OCF thermostat's temperature reading; in this case, the air conditioner would act as an OCF client, sending a Read request for the thermostat's temperature resource. Assuming proper routing from an OCF intermediary, the thermostat would receive this request. Acting as the OCF server, the thermostat would respond to the air conditioner with the data referred to by its temperature resource.

To further standardize the way that entities are represented as resources, the OCF maintains a Common Data Model that defines base schema for all OCF resources and specifically defines requirements and structures for some particular OCF ecosystems. Currently, the Resource Type Specification models "key use cases" that include "Device Control, Notification, Environment Sensing and Control, Energy Management and Energy Saving" [66]. Additionally, the Smart Home Specification discusses aspects of designing products for the home IoT, including mandatory resources for certain kinds of devices [68].

Connecting endpoints takes several steps. First, a new device must undergo "onboarding" to join an OCF network. This process involves joining a network and having a designated "Onboarding Tool" discover the device and perform "Ownership Transfer", which involves setting the device up with proper security credentials and configurations. (The specification does not describe how guest devices might temporarily join a network when they are owned by another network.) Once on the network, there are several protocols that allow endpoints to discover each other. From there, an endpoint can request another's designated discovery resource in order to discover what other resources that endpoint has.

To secure resources, devices authenticate with each other using their security credentials. OCF uses discretionary access control in the form of access control lists (ACLs) to determine what operations an endpoint can perform on a given resource. To secure the communication channel, the OCF relies on the Datagram Transport Layer Security (DTLS) protocol. However, the OCF architecture does not define how to manage credentials or access control lists. It does state that a credential management service (CMS) can be used to create, delete, refresh, issue, or revoke credentials, but the structure and interfaces of CMSs are never defined. Likewise, ACL management is not defined in any detail (although the specification notes that future versions will include more information on ACL management.)

[a]See Roy Fielding's thesis for details on RESTful interactions [27].

Addressing poor security design is especially important at this early stage in the development of the Internet of Things. As industries settle on standardizing architectures and protocols, it is likely that new products will need to continue to adhere to the standards in order to remain interoperable with existing IoT devices. At some point, it becomes financially infeasible to switch to another standard, even if consumers and firms may prefer another technology. At this point of "lock in", the IoT will be stuck with whatever security-related requirements and guidelines those standards have [23].[9] Likewise, successful home IoT devices produced now will be emulated in the future, and it's possible that certain design patterns will normalize. Even without technical lock in, design norms will have a significant influence on the security design of future products.[10] Both for technical lock in and design norms, locking in and normalizing forward-thinking security requirements and guidelines now could positively benefit the security of millions of future IoT devices.

2.2.3 Potential Solutions

Many organizations have put forth a variety of solutions to help developers make IoT devices more secure (through improved security design or otherwise.) Historically, the field of Systems Security Engineering (SSE) was meant to address these shortcomings. SSE as a field aims to "prevent the introduction of vulnerabilities prior to release, rather than patching vulnerabilities afterwards." In recent years, the SSE field has

[9]This has occurred with numerous Internet standards. For instance, people are more or less locked in to trusting the certificates issued by Certificate Authorities, which validate the identities of web sites, despite myriad security problems with the Certificate Authority system [17].

[10]For comparison, many applications use personal questions for password recovery, even though many of these systems are less secure than the passwords themselves.

grown in popularity as engineers have found both that testing and patching alone are insufficient and that investing in security, especially early in the design phase, can provide major returns on investment later on [39]. However, as evidenced by rampant existing IoT vulnerabilities, either SSE processes have not been as helpful as hoped, or developers have not employed them. For example, the National Institute of Standards and Technology (NIST) recently published the NIST Special Publication 800–160, meant to serve as a foundation for engineers in firms wanting to carry out SSE [63]. Largely motivated by recent security breaches, this publication draws from ISO, IEC, and IEEE standards, as well as relevant academic literature, in an attempt to provide engineers with a timely guide to incorporating SSE into their work.

Other groups have eschewed formal engineering processes in favor of publishing checklists of so-called best practices meant to help developers avoid the most common pitfalls in software and IoT development. For example, the Department of Homeland Security has published broad "Strategic Principles" that gives IoT stakeholders a variety of security practices to consider when developing, manufacturing, and administering IoT devices [88]. Separately, the Linux Foundation has released their own list of best security practices through the Core Infrastructure Initiative's Best Practices Badge Program. This program applies to all Free/Libre Open Source Software (FLOSS), including FLOSS IoT projects, and provides a wide variety of development practices to meet in order to earn the project a Best Practices Badge [20].

The Cyber Independent Testing Lab (CITL) is developing a new approach to helping developers test software and IoT security. Instead of providing guidelines for developers and organizations to follow in the process of developing an IoT product, they aim to use quantitative metrics to produce a normalized security score for binaries on a particular platform. These scores will ideally allow consumers to consider security when choosing between products (simiarly to how Consumer Reports helps car buyers consider vehicle reliability.) In turn, vendors will have an incentive to seek to maximize their CITL score by designing their product to perform well under the CITL testing system [91].

All of the aforementioned solutions are "process-based", meaning that they ideally help developers to design more secure products by somehow guiding or structuring the process of developing the product. SSE processes lay out the high level tasks for developers to accomplish. Checklists posit best practices and encourage developers to complete some or all of the relevant ones. CITL's tests encourage developers to tailor their designs to maximize the indicators of security that CITL tests measure. Thus, all of these solutions aim to help by focusing or guiding the use of developers' time.

A series of informal interviews have found that these process-based frameworks have caught on among Chief Information Security Officers (CISOs) as they convince other executives to invest in security.[11] However, it's not clear how much these frameworks actually help. Since the benefits and costs of these processes are never estimated, it is difficult to compare them, or even to determine whether they are doing any good at all [61].

"Outcome-based" solutions present a theoretical alternative to process-based solutions. These sorts of solutions measure outcomes related to security to guide decision making. For instance, they may try to estimate return on investment (ROI) on investments in security, or they may rely on some sort of cost-benefit analysis (CBA) when weighing decisions about aspects of security. However, this approach is currently infeasible to use in practice. Information and computer security risk is notoriously hard to estimate, and predicting how much a given security mechanism will affect risk is even more difficult [38, 61]. Researchers have proposed highly theoretic frameworks [43], others have proposed somewhat more practical frameworks [12], and others have demonstrated the approach in single applications [70], but there none have overcome the difficulties of estimating risk and risk mitigation in any general and practical way. As a result, there is no outcome-based solution in widespread use [61].

It is beyond the scope of this work to survey the field of existing security guidance

[11]Technically, the interviews focused on firms intending to manage risk rather than firms developing products. Despite these differences in domains, the underlying motivation to structure a team's approach to security is similar for both infrastructure risk management and product security design.

comprehensively,[12] but it has covered some of the most currently prominent examples. The field of systems security engineering as a whole is surely the most formal body of literature relevant to guiding developers through security design.[13] Beyond that, there are many varieties of best-practices checklists published by government agencies, industry organizations, and even ad hoc blog posts written by individuals. The Cyber Independent Testing Lab represents a new style of process-based security guidance focused on automated binary testing that is meant to mirror Consumer Reports safety ratings. While it is possible that these frameworks are all helpful, all these process-based approaches are hindered by the current lack of an ability to estimate how helpful (or counterproductive) they actually are. At the other extreme, attempts to quantify the effects of various security mechanisms have been theoretically proposed but have proven difficult to practice at scale.

2.3 Conclusions

IoT security should be reasonable, ideally minimizing risk so as to maximize the overall value of a system. This ideal varies with context, as different stakeholders will perceive risk and value differently, the value of relevant assets determines the appropriate levels of defense investment, and the environment surrounding an IoT system defines the potential threats to those assets. Inevitably, tradeoffs between security mitigation and product features, costs, and usability arise, and the optimal balance of these tradeoffs varies greatly with these situational factors. The singular, universal, or nominally objective ways in which security is traditionally defined are thus incorrect. While these incorrect definitions are occasionally useful as heuristics, a substantial literature exists that demonstrates the perils of ultimately prioritizing security folk wisdom over careful consideration of costs and benefits in context.

Improving security design lies at the heart of making the IoT more reasonably

[12]Security researcher and writer Bruce Schneier has started such a survey with a collection of IoT security and privacy guidelines that he posted on his blog [77].

[13]The technical security field is, of course, larger. However, academic technical security papers in themselves generally deal with narrow single points and are not usually meant as guidance for developers in themselves.

secure. As current exploits show, the high complexity and uncertainty inherent to designing IoT products makes IoT security design stand apart from traditional security design. Although there are other aspects of IoT security, there is little indication that they are as pressing at this time as security design. That architectural standardization and design normalization have not yet occurred makes improving IoT security design even more urgent, as standards and norms could prove to be key avenues through which to bolster (or hamper) the security designs of millions of future IoT devices.

Many organizations have put forth frameworks to aid developers as they create IoT security designs, but they do not appear to help enough. Process-based security frameworks are commonly used, but there is no standard with which to compare them, let alone determine their actual impact on IoT security. Outcome-based security frameworks theoretically measure their own impact, but they remain impractical for real-world use. To make clear forward progress towards reasonable security for the IoT, then, the field must develop methods of gauging the impact of process-based security frameworks.

Chapter 3

Evaluating Security Guidance

In Chapter 2, we discussed how there is no established way to rigorously gauge the value added by process-based security frameworks. While a precise model for estimating the impact of a given framework on the bottom line—for either a single stakeholder or for society—remains elusive, this chapter derives the criteria that are relevant to this evaluation from first principles. Further, it demonstrates the utility of these criteria through the evaluations of several prominent security frameworks.

To be precise, *process-based security frameworks* include any structured guidance with the goal of improving developers' abilities to design or implement[1] information processing systems with reasonable security. These frameworks must focus on particular aspects of a product's security design or on the process of developing a security design itself. To qualify as "structured guidance", a process must be written with the clear purpose of actually helping to guide developers.[2] Although they do not explicitly measure security outcomes, the goal of process-based security frameworks is still ultimately to add value by enabling or encouraging developers to settle upon

[1]Although this thesis emphasizes the importance of security design, these criteria do not require an exclusive focus on design. Not all security frameworks draw this same distinction between design and implementation, and there is no immediately obvious benefit to distinguishing between the two for the purposes of this criteria.

[2]For contrast, outcome-based approaches focus on some metric of value added by security; particular security policies, mechanisms, and development processes do not contribute to outcome-based approaches aside from their ultimate effect on the measurement on added value. Likewise, an academic research paper proposing a new technical security mechanism typically is not written directly for developers, so it would not count as a process-based security framework.

or implement more reasonable security designs at a worthwhile cost.

3.1 Evaluation Criteria

The added value of these frameworks can be gauged from two perspectives. First, a framework can be evaluated in its benefits and costs within a single application of the framework, whether in general expectation or in the case of a particular development team's use of a framework. Second, the benefits and costs of a framework can be extrapolated to estimate its overall effect on society. This section identifies criteria for gauging both perspectives: three criteria capture the expected benefits, versatility of those benefits, and the costs of individual applications of a framework, and three more criteria capture the dimensions relevant to determining the potential domain of a framework as well as the likelihood that endeavors within that domain will actually use it. For each criteria, in addition to examining the criteria itself, we discuss strategies for measuring or evaluating it.

3.1.1 Quality of Guidance

The first and most fundamental criteria to look at is the *quality of guidance* itself. This criteria answers the question, "To what degree does the content of the guidance improve (or hinder) the development of reasonable security designs?" Relevant factors to consider when addressing this question include:

1. *Correctness*: how factually accurate the guidance in the framework is.

2. *Reasonableness*: how well the guidance encourages more reasonable security mechanisms over less reasonable ones in a given situation. Alternatively, if the framework focuses on development practices instead of particular security mechanisms, then this factor covers how well the guidance encourages practices that foster more reasonable security over practices that do not.

3. *Timeliness*: how relevant the guidance is in the current time. To gauge timeliness, it may also be worth noting any systems in place that will adapt a

framework to new information over time.

4. *Completeness*: how comprehensively the framework covers guidance within its scope.

5. *Imbalance*: how a framework leads developers to neglect important aspects of security. For example, a framework could be correct, reasonable, timely, and complete for the guidance in its scope, yet ultimately lead to flawed security designs by giving the impression that out-of-scope aspects of security are not worth consideration.

Evaluation of quality of guidance requires information security expertise and the use of the latest methods in the information security field. Further, it must be performed according to the security situations that experts have identified. This field is large and dynamic, yet, in a sense, its purpose is to determine the quality of security guidance. It would be infeasible to try to reduce the entire field to a single metric for estimating quality of guidance; even if it could be done at a point in time, the metric would have to change every time a new security paper moves the field forward.

3.1.2 Flexibility

The next criteria is *flexibility*, which captures the degree to which developers can tailor the framework to their particular situation. Frameworks fall into a spectrum that ranges from the complete flexibility of no guidance to the complete rigidity, or lack of flexibility, of having to develop a product a certain way, without any options given to developers. Care is necessary to extract realized flexibility from intended flexibility; if a framework states that it can be adapted to situational needs but does not provide the necessary resources and guidance for nontrivial adaptations, then it is not flexible. Between quality of guidance and flexibility, these first two criteria comprise the benefits that a framework can bring about; the former criteria reflects current expected (or measured) benefits to well-understood situations,[3] and the latter

[3]For new and poorly understood situations, a quality of guidance evaluation could not be more than unjustified speculation.

reflects how effectively a framework can adapt to changes and uncertainty.

There exists tradeoffs between flexibility and rigidity that must be balanced. Flexibility gives developers agency to cater their product to the particular stakeholders, assets, and environments that they anticipate, and it grants them the freedom to choose the security mechanisms that they predict is best. Flexibility allows developers to optimize for reasonable security in a variety of situations. However, it also puts the onus on the developers to spend time and resources weighing security mechanisms, leaving room for errors in judgement. Rigidity has the opposite advantages and disadvantages. Rigidity alleviates the burden on developers to weigh potential security mechanisms and decreases room for errors in judgement. However, guidance has limits of specificity; it is not practical to anticipate any and every product that a developer could want to build and specify a perfectly reasonable security design for it. A framework that is too rigid may suboptimally fit a product that it is applied to. Thus, the ideal amount of flexibility adapts to unanticipated products and situations and minimizes developer ignorance and error.

The right balance between flexibility and rigidity depends on the stability of the domain to which a framework is being applied. Loosely speaking, a domain is "stable" when systems in it tend not to significantly differ in their designs from existing systems, and the environments those systems operate in tend to be well-understood and slowly changing.[4] Products developed in fields of high uncertainty call for more flexibility, and, as a field stabilizes, greater rigidity is justified. When there is high uncertainty, there is low likelihood that existing evidence and reasoning about reasonable security will generalize well to new situations, so developers will need to take on more of the burden identifying and weighing prospective security mechanisms. As a field stabilizes and uncertainty decreases, the balance shifts as individual ignorance and errors in judgement become a greater security concern than poorly fitting evidence and reasoning. We hypothesize that the home Internet of Things is currently in a highly uncertain, complex, and dynamic state, and that more flexible frameworks

[4]For instance, the phone market in 2017 is significantly more stable than it was just after the release of the iPhone, whereas the advent of deep learning has probably made the field of artificial intelligence become less stable over the same time period.

are currently appropriate. As the industry stabilizes, security frameworks for home IoT should be able to accommodate more rigid guidance.

We are not aware of other literature that investigates the flexibility-rigidity tradeoff for security frameworks. In addition to proposing this tradeoff, this work demonstrates the qualitative flexibility evaluation of multiple prominent existing security frameworks in Section 3.2. Additionally, it qualitatively evaluates the flexibility of PRSD, the novel security framework proposed in this work, in Chapter 5. These examples show that qualitative evaluation of flexibility can lead to useful insights. It's possible that more precise concepts and quantified methods of evaluating flexibility and stability could lead to even more beneficial insights. Such methods should be explored in future work.

3.1.3 Administrative Costs

If quality of guidance and flexibility comprise the dimensions of a framework's benefits, then *administrative costs* comprises the costs of using it. This criteria is the overhead cost of applying a given security framework to a project less the cost of developing the product without the framework. No matter how high quality and appropriately flexible a security framework is, it is not useful if it is unaffordable or if it costs more than the value it produces.[5]

Evaluation metrics can be designed with existing economic concepts. The general approach is to break down the potential costs of a framework into all the distinct categories through which those costs are manifested. These categories include, for instance, the cost of preparing the overhead documentation required by the framework, the research and development costs of actually carrying out the tasks required by the framework, the costs of hiring and training the developers or other employees who will carry out the framework, certification costs, and the foregone revenues that are estimated to result from a longer development cycle. For each of these categories, the cost

[5]Depending on one's perspective of economic fairness, there is a case to be made that a security framework that provides marginal benefits to users of the finished product could still be worthwhile even if the aggregate benefit is less than the cost to the company of practicing the security framework. Considering this case, while important, is outside the scope of this work.

should be estimated or measured for both the application of the framework and not using a framework. The total administrative costs, then, is the sum of the differences between the framework costs and no-frameworks costs across all cost categories.[6]

Generalizing the administrative costs of a given framework beyond any individual project requires estimating these costs with respect to an arbitrary IoT product. For instance, one might estimate that the certification costs required by a framework are a certain relatively constant cost regardless of the project but that the documentation required throughout development becomes exorbitantly expensive for more complex projects. Such estimations are currently neither trivial nor precisely quantifiable, yet they can still occasionally be valuable. In Section 3.2, we draw conclusions from estimating the administrative costs of multiple existing frameworks. Further, Chapter 5 qualitatively evaluates the administrative costs of PRSD, comparing these estimates to the estimations for the existing frameworks. While these sections show the use of qualitative administrative cost estimation, future work making these evaluations more rigorous could lead to more useful insights.

3.1.4 Scale and Scope

The fourth and fifth evaluation criteria, *scale* and *scope*, relate to aggregate societal value. *Scale* is the range of project sizes to which a security framework can be effectively applied. While some frameworks may work equally well for projects of all sizes, others may grow exponentially more complex with the size of a project and thus be unsuitable for large projects, while others still may have too much required overhead to be useful for smaller projects.

To estimate scale, one must gauge how the work required by a framework varies with the size of a project. Project size itself is a loose concept, incorporating heterogeneous concepts such as team size, lines of code, budget, and system complexity. Different project sizes will make different demands of a given sort of framework; the framework fits a project's scale if and only if it meets all those demands in a way that

[6]Technically, a framework could have negative administrative costs by reducing the work it takes to develop a given product.

is feasible for the relevant teams to carry out.

Scope, on the other hand, refers to the domain of product functions and situations to which the security framework applies. For instance, a security framework with a small scope may only apply to implementing a particular technology into a product or to building a certain kind of home IoT product, whereas a security framework meant to apply to every aspect of the IoT as a whole would have an enormous scope.

Scope is often trivial to evaluate, as it is usually explicitly advertised within a framework's documentation. To be sure, evaluators should take care to ensure that a framework actually follows through on its advertised scope. However, frameworks that do not do so can be handled on a case by case basis. In these situations, evaluators should distinguish between poor quality guidance, in which the guidance is simply incorrect or incomplete, inappropriate flexibility, in which the guidance does not extend to a new or uncertain situation well, and fraudulent scope, in which a framework simply does not cover territory that it claims to cover.

3.1.5 Incentives

The final evaluation criteria, *incentives*, comprises the motivating factors that firms face to use or not use a given process-based security framework. For comparison, while scale and scope evaluations reflect how many projects a framework could potentially be applied to, incentives evaluation helps to estimate how many development teams might actually choose to use a given framework. Put another way, incentives reflects the likelihood that an arbitrary development team whose project falls within the scale and scope of a security framework will choose to use it, given knowledge that the framework exists.

Many factors can incentivize a firm one way or the other. If a firm believes that a framework will provide a worthwhile return on investment,[7] that in itself could provide

[7]A firm can have a return on security investments by preventing attacks that it would be liable for or that would tarnish its reputation and hurt future sales. Recent research has found that about 11% of customers stop doing business with a firm after a security breach [1]. Furthermore, Kwon found that, after several security breaches, people are even willing to switch healthcare providers [53]. Additionally, a lawsuit filed in the wake of the Target security breach has introduced the real possibility that shareholders could hold corporate management personally liable for security breaches [54].

enough incentive to use a security framework. Additionally, an attractive standard could require adherence to a security framework, so the benefits of adhering to the standard could entice developers to use a framework.[8] Moreover, a law requiring the use of a security framework could greatly incentivize its use. Even without being explicitly required by law, a firm may still want to use a security framework to avoid legal liability.[9]

For the most part, one can evaluate the incentives that firms face with existing economic tools. Ross Anderson and Tyler Moore have surveyed the economics of information security [60], which will likely be relevant to many security frameworks. Additionally, one may consider how firms may want to advertise their use of a framework as a market signal [82]. In addition to these tools, current law and legal precedent should be considered.

3.1.6 Criteria Summary

Table 3.1 contains a brief summary of each of the six criteria.

3.2 Evaluation of Several Security Frameworks

To demonstrate the use of these criteria, we use them to evaluate multiple process-based security frameworks, including the NIST 800–160, the Cyber Independent Testing Lab (CITL), and the CII Best Practices Badge Program. As these demonstrations show, although the metrics for evaluating for each criteria are not formally standardized, the criteria are still useful for comparing vastly different frameworks.

[8]Knieps has written a useful introduction to the economics of standardization, which covers this topic [51].

[9]Ever since the FTC successfully brought Wyndham Worldwide Corporation to court for having “deficient cybersecurity” [87], there has been a precedent for the FTC to bring similar administrative action against other firms. Firms may believe that using a framework will help them avoid an FTC complaint.

Table 3.1: Criteria for evaluating process-based security frameworks

Criteria	Description	Evaluation procedure
Quality of guidance	Expected or measured benefits of framework to well-understood situations	Information security expertise
Flexibility	Appropriate adaptability of a framework to changes and uncertainty	Qualitative evaluation with respect to stability of framework's domain
Administrative costs	The costs of using a framework over the costs of using no framework	Summing of the differences between framework costs and no-framework costs across all cost categories
Scale	Range of project sizes to which a framework can be effectively applied	Estimation of work required by framework as function of various factors of project size
Scope	Domain of product functions and situations to which framework applies	Check framework documentation
Incentives	Motivating factors for a firm to use or not use the framework	Existing tools from economics and law

3.2.1 NIST 800–160

3.2.1.1 Introduction

HI In 2016, the National Institute for Standards and Technology published Special Publication 800–160 *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems* [63]. This publication serves as a "collection of related processes" (page xi) to aid in engineering any general system for security throughout its entire life cycle. First, the publication discusses its motivation and conceptual framework, laying out definitions for concepts such as *trustworthiness*, *security*, etc. Next, the publication details each of its thirty processes intended to aid in security engineering. NIST 800–160 is "intended to be extremely flexible" and is emphatically not meant to prescribe exactly how every organization must perform security engineering (page x). The publication describes how each process could theoretically be tailored to meet the needs of the organization applying the process, although hardly any guidance for actually tailoring the processes is provided.

Several concept definitions from the publication are worth mentioning here. To start, it notes that *trustworthiness* is the state of being "worthy of being trusted to fulfill . . . critical requirements" of the system. It goes to assert that "measures of trustworthiness are meaningful only to the extent that the requirements are sufficiently complete and well-defined, and can be accurately assessed" (page 1). Later, it further assumes that there is an objective and universal way to determine trustworthiness, and that a system can in fact be "trustworthy in general" (page 8).[10]

The publication defines *security* as the "freedom from those conditions" that could cause harm to stakeholder assets (page 2). Moreover, it argues that security engineering should strive for *adequate security*, a concept similar to this work's *reasonable security*, as it entails deciding on a "reasoned sum" of security mechanisms that balance the needs of all stakeholders (page 16).

[10]As described in Section 2, these assumptions are incorrect, because trustworthiness only makes sense when it is respective of a specific stakeholder's perspective.

After laying out the groundwork, each of the thirty processes are detailed. Each process comprises a description of that processes' purpose, the *outcomes*, including the results and deliverables, that the execution of a process produces, and the activities or tasks that the process entails performing. The descriptions of activities and tasks includes details on what the process entails, but provides little guidance for how to actually carry them out. They also include references to relevant standards (usually ISO standards). No technical how-to guides are referenced, as this publication is intended for systems in general.

Taken together, the processes lead development teams through several overarching tasks. First, developers identify all stakeholder requirements and determine which ones to deem "security requirements", which abstractly detail what stakeholders will want to get out of the system. Then, developers determine how to translate those stakeholder requirements into "system security requirements", which detail particular aspects of the system that will fulfill those stakeholder security requirements. Next, developers and engineers actually make the system. After (or during) the process of building the system, the development team performs validation and verification to make sure that the right system was built and that the system was built right. Finally, other processes such as "Human Resource Management" and "Quality Assurance" support these tasks one way or another.

The publication at times briefly acknowledges that tradeoffs between stakeholder or system requirements may exist, but it does not deal with managing those tradeoffs. Likewise, it briefly notes at the end of its last appendix that its possible for security features or even the execution of the processes in the publication to cost more than the benefit they provide. It does not explain in any detail how to deal with this possibility.

3.2.1.2 Evaluation

Since the purpose of NIST 800–160 is to help development teams organize and systematically cover all security-relevant aspects of engineering for a system, its *quality of guidance* rests on the effect it has on the structure of an organization as well as how thorough it is in systematically covering its subject.

As a result, the overall benefit of this publication is bounded by the security expertise of the development team using it, as the publication does nothing to augment security expertise in itself. That said, because the thirty processes cover so many aspects of security engineering, teams that follow these processes are likely to examine aspects of security engineering that they would not otherwise consider, so it is useful in that regard.

It does have some correctness issues. Section 2 discusses how security is relative to particulars of stakeholder perspective, assets, and environmental context, so the publication's assertion that trustworthiness can be universal and objective is unfounded. Conclusions based on this assumption throughout the publication may likewise be false.

This publication almost precludes reasonableness. Despite acknowledging that tradeoffs exist, its requirements-based structure and processes leave very little room for dealing with them. Making developers determine security requirements from the outset entails making nearly arbitrary judgements about what security mechanisms to include or not include, so there is no room to weigh competing requirements and rejected requirements against each other throughout product development. Tailoring the processes to incorporate weighing between tradeoffs would entail fundamental restructuring. Thus, this publication relies on the development team identifying stakeholder and system requirements from the outset that happen to lead to reasonable security.

NIST 800–160 is also a bit imbalanced, as the processes do not lead developers to account for interactions with parts of the system that are unforeseeable and outside the development team's control. This is particularly detrimental to home IoT networks; coordinated automation—one of the key potential benefits of the IoT—relies upon interoperability between arbitrary products that may be designed by different firms. While not technically in error, the guidance in this publication may lead development teams to plan for such situations poorly, as it often assumes that the engineers have full control over the system.

This publication is timely. It was released in late 2016 and refers to the latest literature. It likely will not go out of date quickly.

In all, NIST 800–160 has some positive benefits but also some serious shortcomings in its guidance, so its overall quality of guidance is mixed.

NIST 800–160 is intended to be highly *flexible*, although its realized flexibility is mixed. While all processes could theoretically be tailored, it may not be clear how to best tailor a process to a firm's needs, especially because not all processes clearly justify why they are structured the way that they are structured. There are also aspects of this framework's design that are highly rigid. Perhaps most notably, many of its processes rest upon a requirements-based approach that make it difficult to consider competing tradeoffs throughout the design process. Requirements entail drawing a sometimes-arbitrary binary line between what is needed and what is not. Yet, there may be some desirable features that do not make the cut as requirements and thus are not considered later, and other requirements may subsequently be given more priority and effort than they deserve. This approach might be fine for some stable systems, but it does not suit engineering for highly uncertain and unstable systems like the IoT. Thus, NIST 800–160 is probably too inflexible for many IoT systems.

It's difficult to predict this framework's *administrative costs*, as it has not been applied in practice, so there is no empirical evidence one way or the other. Many processes do require analysis and documentation that clearly take considerable effort, but it is not clear how much of that effort firms already tend to do in the first place.

Turning to aggregate criteria, we first consider the NIST framework's potential *scale*. The highly tailorable processes contained in this framework could apply to a large range of project sizes. That said, many of its processes (e.g., "Portfolio Management") really only apply to larger projects with more complicated bureaucracies to support them, so larger projects may benefit most from this framework as a whole. Thus, NIST 800–160 probably scales somewhat well, but is best suited for larger undertakings.

As for *scope*, this publication explicitly covers any system that security engineering may apply to. A quick look through its contents clearly shows how committed NIST was to this generality. For these systems, the publication focuses on structuring how the organization deals with security.

An organization's *incentive* to use this framework rests entirely upon whether or not its managers believe its benefits are worth its administrative costs. It seems that many organizations will try it; as an institution, NIST is widely respected, and many CISOs rely on another NIST framework to justify cyber security investments. CISOs tend to share information with each other, so the publication will likely develop a reputation over time [61]. Thus, as with administrative costs, uncertainty over the incentives to use this framework may be too large to justify a concrete prediction at this time.

To summarize, NIST 800–160 has a mixed *quality of guidance*, too little *flexibility* for the home IoT, and highly uncertain *administrative costs.* Its *scale* covers a huge variety of project sizes, and its *scope* includes organizational structuring for any systems security engineering endeavor, but it is too early to say whether or not the *incentives* are there to get organizations to use this security framework.

Even though the evaluations for administrative costs and incentives are too early to complete at this time, the mechanisms identified in the evaluations bound what the results could be and provide indicators that will help to complete the evaluation in the future. In these ways, their evaluation is still helpful.

3.2.2 Cyber Independent Testing Lab

3.2.2.1 Introduction

The Cyber Independent Testing Lab, run by infamous hackers Peiter and Sarah Zatko, is developing and using a software tool that, given a batch of executable binaries for a particular platform[11], will assign security scores to each of the binaries on the platform. These scores are normalized relative only to other tested files on the same platform [91]. This tool reflects a definition of security based on the difficulty of

[11]Thus far, desktop operating systems are the only supported platforms. IoT platforms will be supported in the future. Security scores for binaries on different platforms are not comparable.

finding exploits that enable privilege escalation or program disruption. Accordingly, the security scores indicate a relative prediction of that difficulty between binaries on a platform.

The authors explicitly have a narrow security scope; they do not include, among other things, application configuration, "past history" of a binary, interpreted scripts, and corporate policies like update policies. Additionally, they also implicitly exclude many security design flaws and human factor vulnerabilities. Rather, they focus almost exclusively on implementation bug vulnerabilities.

The CITL tool currently focuses on several factors gleaned from static analysis of binaries.[12] Many of these factors have been informally publicized, and CITL plans to formally publicize them all when they are finalized. Additionally, future versions of the tool will incorporate factors from dynamic analysis into security scores. The tool only needs executable binaries of applications to measure scores; CITL is interested in the actual programs, not the intended ones. The tool is low cost; the Zatkos claim that it can analyze about 100,000 binaries in 4 days.

CITL will operate on a business model akin to Consumer Reports. They want to publish security estimations so that allow customers to make informed decisions about product security when choosing between applications. CITL is a 501(c)3 organization.

3.2.2.2 Evaluation

CITL looks at difficulty of gaining privilege escalation or disrupting program execution by exploiting implementation bugs, so its *quality of guidance* reflects how well it can lead developers to implement more reasonably secure products. Its guidance is the indicators of security that it measures.

The Zatkos are renowned technical security experts with decades of experience, so they have an immense understanding of indicators of security. While they will probably use this experience to design their tests well, their work has yet to be published and analyzed for correctness.

[12]Static analysis is analysis of the binary contents and metadata of an executable, whereas dynamic analysis is any analysis of an executable while it executes.

CITL gives the burden of determining reasonableness to consumers. The security concept it measures deals with predicted difficulty to exploit a binary; while it does chastise programs for having lower security scores, it is up to users to determine how reasonable the tradeoffs between this aspect of security and other factors are. This is comparable to Consumer Reports, where it's expected that consumers will weigh the reliability of a car against its price and other desirable features on their own. It is unclear how well equipped consumers are to understand the security risks they take and the results of CITL tests when they make their purchasing decisions.

CITL is timely, as it is currently under constant development to keep it up to date with relevant platforms and security threats.

If CITL grows large enough, it could potentially push companies towards thinking that implementation bugs, and specifically the aspects of bugs that CITL tests measure, are the only important aspects of security. This imbalance could cause developers to try to game CITL at the expense of other factors of security. As currently presented, CITL as an organization is clear about its limitations, so this caution is merely a warning of what to watch out for in the future rather than a current shortcoming.

CITL as a tool is highly rigid and *inflexible*. Given a platform and some executables, the resulting security scores are deterministic, so organizations cannot tailor it for their needs. This is intentional. The factors that CITL measures are proven indicators of vulnerability that are unlikely to change quickly, so this rigidity is justified for the currently supported platforms. It is currently unclear how IoT implementation bugs might differ for home IoT from traditional desktop applications, but there is currently little reason to expect much to change. Further, CITL could quickly adapt its IoT platform tests if unique IoT attack vectors are found. Thus, CITL currently has justifiably low flexibility.

The *administrative costs* of CITL testing is minimal, as the lab can automatically run many thousands of tests per day, and any person could easily run these tests themselves to verify the results. Furthermore, the lab's security experts take on the responsibility of broadly interpreting test results for the public. Interpretation takes much more manual labor than the automated tests, but there is no reason to doubt that they will be able to scale up to the level of Consumer Reports.

For developers looking to use CITL for guidance, the added costs of designing for these factors is completely optional—organizations are free to choose how much time and energy to spend learning about the CITL test indicators and tuning their products to score well on them. As a result, while attaining a nearly perfect security score may be out of reach for some products, all organizations can choose to use CITL guidance as much as their budget allows.

Moving to the aggregate criteria, we examine the *scale* of CITL testing. It's rather trivial to see that CITL applies equally well to projects of all sizes. Any project with executable binaries can quickly be tested, no matter how much work goes into producing those binaries.

Separately, CITL has a large yet limited *scope*. It currently only applies to executable binaries on Linux, Mac, and Windows. "IoT" will eventually be included, but it is unclear how they will categorize IoT platforms or how comprehensive those platforms will be.

The actual *incentives* for CITL can be split between those performing the tests and those using the tests. The lab itself does the testing, and they have a strong incentive to test as many programs as possible, because tests are cheap and they want to serve the largest possible customer base. Additionally, the people in the lab have a strong incentive to design and run trustworthy tests, as their entire business rests on building a strong reputation and earning consumer and industry trust.

At this time, it is unclear what incentives will motivate developers to use (or not use) the CITL security indicators for guidance. It is possible that the CITL framework could earn a strong reputation as a signal for security. Consumers may start to make purchasing decisions that incorporate CITL security scores, which would incentivize developers to write programs that achieve higher security scores. If this signal never materializes, then the incentive will simply reflect the existing incentive to make secure products, combined with the developers' faith in the CITL test's guidance.

To put it shortly, CITL has a high but narrow *quality of guidance*, has justifiably low *flexibility*, and has minimal mandatory *administrative costs.* Its *scale* covers projects of all sizes, and its *scope* covers many traditional computing applications, but it is unclear how much of the home IoT it will cover. The lab's team has a strong incentive to produce great tests, but it will only incentivize better security practices if consumers start demanding high security scores from vendors.

3.2.3 CII Best Practices Badge

3.2.3.1 Introduction

Sponsored by the Linux Foundation, the Core Infrastructures Initiative (CII) Best Practices Badge Program is a system for FLOSS projects to self-certify that they follow a variety of security best practices that have been identified by the CII [20]. CII also provides a web application for projects to publish evidence that they meet some or all of the best practices in the list. Upon showing that they meet a sufficient amount of the criteria, projects can receive a badge they can publicly display that shows how well they meet the criteria.

The purposes of this project are to encourage projects to follow the best practices, help new projects learn what these best practices are, and enable users to make more informed decisions about how well various FLOSS projects conform to the best practices, which ideally indicates better-written, more secure code.

Currently, participating projects either have the badge or not; eventually, CII intends to release more advanced badges that can be attained by conforming to more stringent best practices requirements. With these more advanced badges, more advanced projects can demonstrate that they go above and beyond the minimum best practices requirements.

It's not entirely transparent why some badge criteria are chosen and formulated the way they are. Some criteria include explanatory rationale, but others are presented as truisms without justification. Further, the authors acknowledge that no set of best practices are guaranteed to lead to perfect security, and they do not strive for this. They merely want to encourage general practices that, if followed, would almost always elevate a project to more reasonable security.

CII aims to help as many projects as possible. While there are some more advanced optional best practices that are only feasible for larger projects, all the ones required to earn a badge are restricted to activities that would be feasible for a single-person project to carry out. Furthermore, care is taken to have best practices that are as general as possible. They never require, among other things, any particular technology, platform, or programming language. By sticking to these principles, CII hopes to have this program remain relevant even as technologies and development patterns change.

3.2.3.2 Evaluation

To judge the *quality of guidance* of this Best Practices Program, one needs to discern the degree to which following these best practices will lead to more reasonable security than ad hoc development.

Currently, it appears that all of the best practices listed are correct insofar as they are not factually incorrect or outright counterproductive. The guidance is also timely, as the project is actively maintained and is designed to be independent of any sort of technology that could go obsolete in the foreseeable future.

This guidance has some reasonableness problems. Most of these best practices are chosen because they are the "lowest common denominators" that are almost always smart practices, so following them will often lead to more reasonable security. That said, there is very little guidance about why best practices are good practices or situations where they may be counterproductive.[13]

Completeness is the largest shortcoming here. While the best practices touch on many important aspects of security, they are rarely comprehensive. Further, in trying to cater to the lowest common denominator, there are many practices that are only occasionally beneficial that the authors explicitly decide not to mention.[14] They

[13]For example, their HTTPS advice is dangerously absolute. Troy Hunt, a Microsoft Regional Director, has discussed the problems with this specific kind of advice [85].

[14]For instance, one of the talk pages discusses how "automated regression test suite includes at least one check for rejection of invalid data for each input field" is not included because it would be a "tough burden" that might not be "worth it" for many projects [22]. However, by not including it,

certainly do not include more cutting-edge security techniques that could be helpful but are not as proven.[15] This lack of completeness is not inherently problematic; savvy developers certainly are not stopped from practicing other helpful practices that are not listed. However, it will be difficult for the developers that most need this guidance to know what they are missing. As a result, the guidance in the CII Best Practices Badge Program is usually good about what it does have to say, but developers need to be careful about determining what it does not say.

The structure of this framework also makes it less *flexible* than it should be. Most of the required practices are beneficial most of the time; in this regard, it is usually appropriately rigid.[16] However, the merely suggested and optional practices are not always completely justified, leaving developers little indication about how important it may or may not be to follow said practice. To generalize, by making it difficult for non-experts to judge the importance of various criteria, developers have little ability to cater the guidance in this framework to their project's needs. Although this program attempts to be adaptable to future demands through its technology-agnosticism, this guidance is not adaptable to how future technologies could shift the tradeoffs between aspects of security and other considerations.

This program has manageable *administrative costs* for most projects. While carrying out these best practices certainly takes time, the overhead in actually participating in the best practices badge program is tiny. These low costs are evidenced by the number and variety of projects that have perfect scores in this program: there are dozens of finished projects, including major FLOSS works like Xen, LibreOffice, and dpkg [21].

Turning to aggregate criteria, we first note that the Best Practice Badge Program has a very high *scale*—all of its best practices are carefully designed to apply to any size of FLOSS project, and, as previously discussed, there are many large and small

the projects for which this practice could be worthwhile will not receive the guidance.

[15]For instance, while the framework does say that projects "SHOULD" use perfect forward secrecy where applicable, they never mention, say, homomorphic encryption.

[16]It would be ideal if practices that are always required always had rationale for why they are required, so that advanced developers operating in the corner cases could adjust accordingly. However these situations are probably few and far between.

projects that are fulling passing in this program. Furthermore, it has a limited but large *scope*, as it only applies to FLOSS projects,[17] but it otherwise applies to them without reservation.

Incentives with FLOSS projects are complicated, because their developers do not have a profit motive. Instead, they tend to be motivated by status, especially if the status can signal ability to potential employers. However, if this badge program either became enviable or expected by the FLOSS community, then developers might want to earn their project a Best Practices badge to either earn respect or avoid contempt, respectively [56].

Overall, then, the CII Best Practices Badge Program can provide a positive *quality of guidance* for careful developers, yet it is not *flexible* enough to adapt to many projects optimally. That said, it has very low *administrative costs.* It has a high *scale* and *scope*—applying well to any FLOSS project—although it has yet to develop a strong *incentive* for developers to use it.

3.2.4 Evaluation Discussion

Table 3.2 summarizes the evaluation of these three frameworks.

As this table illustrates, these criteria allow one to compare and contrast vastly different process-based security frameworks, even when more information is needed: One can see that NIST 800–160 may not be worth using for home IoT, even if its administrative costs end up low; CITL, once finished will likely prove valuable to developers and consumers, although it covers only small part of security; the CII Best Practices Badge Program provides some helpful guidance at a low price to FLOSS programs, but it is far from a complete solution.

These criteria also reveal gaps in the guidance available to developers. Most importantly, no framework provides guidance that helps developers weigh tradeoffs as they create and modify security designs. In other words, developers have no assistance in actually making security reasonable. This gap extends beyond the three frameworks

[17]Many of these best practices could potentially help projects that are not free or open source, but such projects are ineligible for badges.

Table 3.2: Summary of evaluations of security frameworks

Criteria	NIST 800–160	CITL	CII Best Practices
Quality of guidance	mixed	high	medium-high
Flexibility	too little	appropriately none	too little
Administrative costs	uncertain	minimal required[a]	very little
Scale	high[b]	maximum	high
Scope	organizational structure of all systems security	traditional executables binaries[c]	all FLOSS projects
Incentives	uncertain	uncertain for developers[d]	uncertain

[a]Additionally, manageable for testers
[b]Biased towards large projects
[c]There are plans to include IoT in the future
[d]Additionally, strong for testers

discussed in detail here—we could not find any process-based security frameworks for the Internet of Things that adequately helps developers weigh the benefits and costs of competing security designs. The academic literature relevant to this weighing has not been translated to a form that is practical for developers to use. Because reasonable security design is such a critical and unique aspect of IoT security, this gap should not be ignored.

These criteria also serve as a foundation for further discussion about how to compare process-based security frameworks. While the criteria and evaluation procedures are useful, further work to make evaluations more precise and standardized would likely lead to even stronger insights.

Chapter 4

A Framework for IoT Security Design

4.1 The Processes for Reasonably Secure Design

In order to deal with the high uncertainty and complexity of Internet of Things security, developers need assistance weighing their options when creating their security designs. Few process-based security frameworks provide room to vary security designs based on the expected stakeholder perspectives, assets, and environments that a device could be used in. Of those that do, we are aware of none that provide the tools that developers need to determine what security designs will be most reasonable for the situations their devices will face.

Here, we propose the Processes for Reasonably Secure Design (PRSD) process-based security framework to help fulfill this need for home IoT devices[1] as they incorporate the Open Connectivity Foundation standards into their designs.[2] This framework helps developers comprehensively and systematically identify and consider the security threats a prospective or existing device may introduce, options for appropriately mitigating those threats, and the tradeoffs between those options. Without undermining their responsibility to use their own judgement to weigh tradeoffs and ultimately determine security designs, PRSD offers them a coherent approach to iden-

[1]We focus on the home IoT space, because it is among the most rapidly developing IoT domains yet suffers from high rates of security vulnerabilities.

[2]We focus on the OCF standards, because they are the open communications standard with the most widespread industry support.

tifying and prioritizing the information pertinent to carrying out that responsibility.

In support of this goal, PRSD is flexible enough to adapt to the changing needs of developers using it while still providing sufficient structure to help avoid errors, and to entail as few overhead tasks as possible. As a result, applying this framework should help developers design their OCF-compliant home IoT products with more reasonable security.

While the complete framework is described in Appendix A, this section describes the most notable features of its design.

PRSD operates on existing external information and results in *security decisions*, which are security mechanisms that the developers have decided to include in the design of a product. More specifically, the *processes* that comprise PRSD are separate but linked analysis operations that apply to their *dependencies*, or input information, and produce conclusions called *outputs*. To follow the framework, developers should carry out the analyses of a process whenever its dependencies are prepared. Outputs to processes are the dependencies of others, so developers can proceed from one process to the next until they have settled upon security decisions.

Understanding that development teams rarely have a complete and final vision for a product at its inception [55], all analyses can be performed with whatever level of detail existing information permits. As changes are introduced, developers can update the existing analyses accordingly. The analyses in PRSD are explicitly itemized and linked in order to make it easy to only reconsider the analyses affected by a change.

PRSD is specifically tailored for dealing with security threats enabled by the OCF architecture on home IoT devices. For instance, PRSD is suited for considering how to protect the OCF communications of a refrigerator and a new vacuum cleaner as it is introduced to one's home OCF network, but it is not suited for reasoning about the physical security of those products or the OCF communications of a networked vehicle. This scope is limited so that PRSD is appropriately flexible to its subjects; more stable aspects of design and market segments deserve more rigid and specific treatment, and less-understood ones require more flexibility.

Additionally, PRSD is limited to the security design of a product. Implementation, which includes translating designs (security or otherwise) to code, compiling the code into executable software, and production of the actual devices, is out of scope. PRSD aims to reveal how a device *as intended* could be exploited to cause harm, so it assumes that implementation is correct and follows best practices. This focus is motivated by the discussion in Chapter 2, where security design was recognized as a current, critical, and unique problem for the Internet of Things.

Together, the structure of this security framework should enable developers to realize more reasonable security designs in their OCF-compliant home IoT products. Every process is geared to help developers to rigorously articulate the potential threats enabled by their prospective products, the security options that may defend against those threats, and the tradeoffs between those options. In doing so, they can ultimately weigh their choices more comprehensively and accurately than they likely could without PRSD. Furthermore, every aspect of the framework is designed to be highly flexible, as is appropriate when designing for the current home Internet of Things. Without sacrificing quality or flexibility, each process has been carefully structured to minimize unnecessary work in order to keep the costs of using this framework low.

4.1.1 Process Outlines

Here, we briefly outline each of the processes in PRSD. To help illustrate, we provide snippets of the case study from Appendix B. In this case study, PRSD is applied by a hypothetical startup called "Lockr" to their plans to build an OCF-connected, phone-operated door lock called "SuperLock" that is tailored for Airbnb hosts and their guests.

The first process, *Situational Modeling*, depends only upon existing external knowledge. Carrying out this process results in outputs that model the potential behavior of a prospective product and the environments it could act in, including the product's capabilities, environment, and the relevant stakeholders. Capabilities include any functions that a device can possibly carry out to make sense of or act upon its environment.[3] The environment includes any relevant details of the surroundings that the developers expect their product to be used in. The relevant stakeholders include any agents that have an interest in or influence over the products besides attackers, who are separately analyzed in the next process. This process is a necessary first step through which developers can systematically consider how a product might interact with the world and what is at stake through those interactions. The outputs of this process are the necessary foundation to considering potential threats.

Table 4.1 shows the SuperLock capabilities, and Table 4.2 shows its stakeholder models. Examples of information from its environmental models include that SuperLocks will be physically accessible indoors and outdoors and that they will guard items of potentially enormous value.

Table 4.1: SuperLock capabilities

Capability	Description
lock	lock and unlock mechanism
communication	Communication over the home's OCF network. These communications are intended to include both lock requests and administration with phones.
update	receive Internet updates over a proprietary protocol
physical lock	backup physical lock and unlock mechanism
reset	reset button restores SuperLock to factory settings

With this model, as well as some general threat expertise, developers may perform the next process, *Threat Modeling*, which outputs security threats enabled by the OCF architecture of the prospective product. These threats include the potential harms that a product could enable, the attackers who might want to cause those harms, and the means by which they could cause them to happen through the aspects of the product relevant to the OCF architecture. The goal of this process is to help developers create a reasoned threat model, rather than have them rely on their intuition and limited working memory later.

[3]Capabilities are separate from a device is meant to do or not do. This exercise intends to help developers determine what a device could possibly do if an actor, such as an attacker, had complete control over the device.

Table 4.2: SuperLock stakeholder Models

Type	Stakeholders	Interests and Assets	Expertise	Resources
Users	homeowners, Airbnb guests	SuperLock, safety, privacy, valuables, OCF network, availability	minimal	minimal
Admins	homeowners	SuperLock, safety, privacy, valuables, OCF network, availability	minimal	low
Developers	Lockr engineers	company revenue	high[a]	full time job
Support	Lockr engineers	company revenue	high[a]	full time job
Third parties	(none)			
Public	People who use the Internet	Internet access, time		

[a]Above average security expertise, compared to other developers.

Table 4.3, Table 4.4, and Table 4.5 list the harms, attackers, and means from the SuperLock case study.

Table 4.3: Potential harms resulting from SuperLocks

Harm	Description	Assets at risk
break in	attackers get into the house	safety, privacy, valuables, company revenue
locked out	users prevented from getting into home	safety, availability, company revenue
damaged	the lock is damaged	safety, privacy, valuables, company revenue
pivot harm	the SuperLock is exploited and used as a proxy to devices on the OCF network	OCF network, company revenue
bot	the SuperLock is incorporated into a botnet	Internet access, time, company revenue

Proceeding through the next process, *Security Option Enumeration*, involves coming up with security options for combating the threats and explicitly laying out the tradeoffs between those options. This option walks developers through the process of considering alternatives for dealing with each means from the threat model, followed

Table 4.4: Potential attackers with an interest in SuperLock harms. Exclamation points after harms indicate relatively higher expected interest in causing that harm, which in turn indicates a greater willingness to put resources towards causing that harm.

Attacker	Description	Harms of interest
thieves	criminals wanting to commit burglary	**break in** (!!), **damaged**
prankster	people looking to cause mischief	**break in**, **locked out**, **damaged**, **pivot harm**
government	agents of law enforcement or espionage	**break in**[a], **damaged**, **pivot harm**[a]
quarreling[b]	users fighting with each other	**locked out**
seizure[b]	landlords or other authorities seizing property	**locked out**
hacktivists	activists who make statements through hacking	**pivot harm** (!), **bot** (!!)
expansive crime	organized crime looking to steal data or amass botnets	**pivot harm** (!), **bot** (!!)

[a]States will have an strong interest in causing this harm to certain suspects or activists, but these cases are extremely rare, and those users will likely know the risks.

[b]It's possible that more harm could result from trying to protect against this attacker.

Table 4.5: Means through which attackers could cause harms by leveraging SuperLock OCF communications.

Means	Description	Harms
spoof admin	impersonating an administrator	**break in**, **locked out**, **damaged**, **pivot harm**, **bot**
admin commands	sending commands that entail administrator-level actions	**break in**, **locked out**, **damaged**, **pivot harm**, **bot**
spoof user	impersonating an authorized user	**break in**, **locked out**
user commands	sending commands that entail user-level actions	**break in**, **locked out**
MITM onboarding	man-in-the-middling the initial onboarding process	**break in**, **locked out**, **damaged**, **pivot harm**, **bot**

by an analysis of their relative merits and disadvantages with regard to security, product features, costs, and usability. Security options do not need to be fully specified at the outset; this process helps developers focus on the most important alternatives first and return to consider competing details of the chosen options later. This structure provides the information developers need to weigh their options with increased rigor, and it feasible to carry out even for complex designs.

For example, some of the security options considered for SuperLock are authentication schemes: **auth/password/vendor** relies on a single password for all authentication of all SuperLocks, **auth/password/user** tasks administrators with setting passwords for administrators, normal permanent users, and temporary guests, **auth/crypto/simple** has SuperLocks manage authentication by generating and managing authentication keys based on public key cryptography, **auth/crypto/local** relies on another OCF device in each home to generate and manage keys, **auth/crypto/global** relies on one global system for key management, and **auth/biometric** uses phones' fingerprint scanners, sending some representation of the fingerprint to the lock for authentication. Table 4.6 provides a heavily abbreviated summary of their relative security effects. Other tradeoffs cover, for example, the large support costs of a global authentication system for **auth/crypto/global** and the low usability of passwords compared to the other options.

Next, the *Architectural Synthesis* process has developers determine how to design each security option within the OCF specification. In doing so, it may turn out that some security options are required, others are not allowed, and others have tradeoffs that are fundamentally altered by the specification. These effects could change which security options are ultimately included in the product's security design.

Table 4.7, for example, covers the aspects of the OCF specification relevant to choosing between the authentication scheme options.

Any competing security options that the architecture does not settle are considered in *Tradeoff Resolution*, the process which results in the security decisions. This analysis involves weighing the expected security benefits (in terms of harms prevented or mitigated) against the other tradeoffs involved in a security option. The process

Table 4.6: Security options for the SuperLock. Dollar signs indicate relatively how many resources needed to overcome a security option.

Option	Means				
	spoof user	**user commands**	**spoof admin**	**admin commands**	**MITM onboarding**
auth/password/vendor	\$[a]	\$[a]	\$[a]	\$[a]	\$[a]
auth/password/user	\$\$	\$\$	\$\$	\$\$	
auth/crypto/simple	\$\$\$\$	\$\$\$\$	\$\$\$\$	\$\$\$\$	
auth/crypto/local	\$\$\$	\$\$\$	\$\$\$	\$\$\$	
auth/crypto/global	\$\$\$[b]	\$\$\$[b]	\$\$\$[b]	\$\$\$[b]	
auth/biometric	\$\$\$	\$\$\$	\$\$\$	\$\$\$	

[a]One exploit leaves all SuperLocks vulnerable.
[b]One exploit leaves all SuperLocks, and potentially many other devices, vulnerable.

Table 4.7: Relevant documentation from the OCF specification and IoTivity, and a description of their impact on the previously identified security options and their tradeoffs. All specification documents are from v1.1.0.

Documentation	Impact
Security 9.3.6	**auth/password** options not allowed
Security 9.3	**auth/biometric** implicitly not allowed
Security 9.3.3	Description of Asymmetric Authentication Key Credentials, suitable for **auth/crypto/simple**, **auth/crypto/local**, and **auth/crypto/global** and supports revocation
Security 9.3.5	Description of Certificate Credentials, suitable for **auth/crypto/local** and **auth/crypto/global** and supports revocation
Security 5.1.1.2	Access Manager Service for **auth/crypto** boosts usability but decreases security (suitable for consideration as a sub-option)
Security 12.1	ACL guidance for **auth** is not specified, but will be in future versions of the specification

helps developers by explaining how each of the outputs of the previous processes contribute towards this end. Final judgement on security decisions is left to the developers.[4]

In the case of the authentication options, we selected **auth/crypto/simple** as a security decision for SuperLocks.

Continually affecting all of the other processes, the *Updates and Maintenance* process enables developers to quickly incorporate new information, design changes, and increased design detail into existing analyses and outputs. As dependencies to processes change, this process describes how to redo relevant analyses in that process with minimal unnecessary effort. The correctness and efficiency of this process are of the utmost priority, as modern software—and, by extension, IoT—engineering is typified by frequently changing requirements and incremental development [55].

For SuperLock, we explored several sub-options of **auth/crypto/simple**—namely, selecting strategies for sharing credentials with guest users and approaches to managing the lock's access control lists—in order to demonstrate how to update analyses for incremental development.

The relationship between these processes is illustrated in Figure 4-1.

4.2 Case Studies summary

To illustrate how the Processes for Reasonably Secure Design works in practice, Appendix B describes its application to four case studies. These case studies serve the dual purpose of providing concrete examples of what PRSD looks like and demonstrates certain aspects of it that give evidence for the claims made about what it achieves.

These case studies are limited to hypothetical situations, as there are no actual certified OCF products to which PRSD could currently be applied, even in hindsight.[5]

[4]More rigidity may be warranted as home IoT design patterns stabilize, but this degree of flexibility is appropriate for the current uncertain and dynamic nature of the home IoT.

[5]There are currently three OCF-certified products [65], but none publish their design details, and they were not willing to discuss their designs with us.

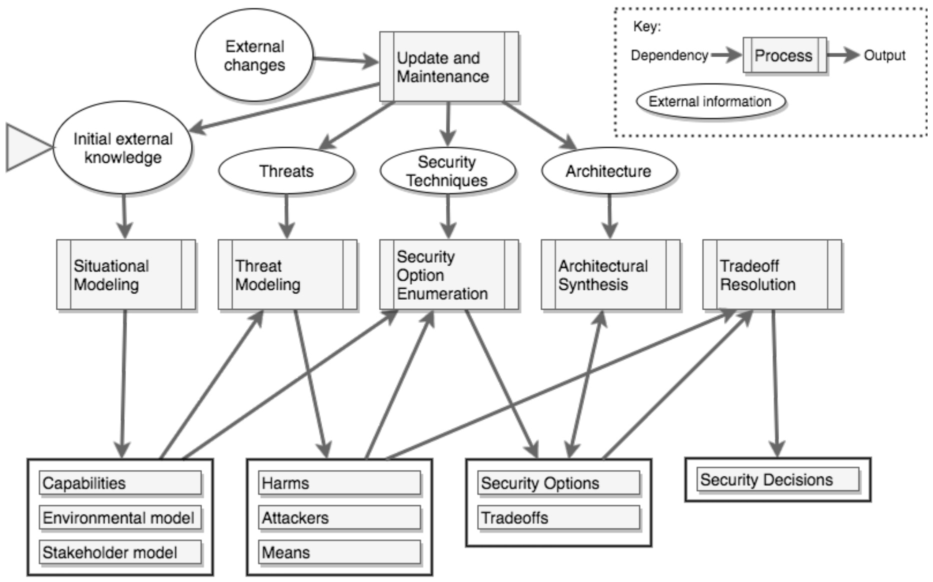

Figure 4-1: Diagram of the Framework

While this does limit the ability of the case studies to demonstrate the efficacy of PRSD, these hypothetical scenarios still provide ample material from which to draw conclusions about the usefulness of PRSD.

The first case study covers "SuperLock", a door lock which can lock or unlock through OCF communications. It exemplifies an IoT product that has simple behaviors but which requires significant security. This case study covers a high level initial application of PRSD, followed by a reapplication of PRSD to some sub-options of chosen security decisions.

The next case study starts off where the previous one finished, but with the company wanting to integrate SuperLocks with Airbnb services to automatically provide door access to Airbnb guests for the duration of their stays. This case study most notably exemplifies how to update existing analyses to accommodate changing design requirements, and it also demonstrates how to apply PRSD at the boundary between the OCF specification and third party proprietary interfaces.

Third, we look at "Hub", which takes voice commands and responds with audio feedback to manage and operate an arbitrary variety of third party devices and cloud services. This case exemplifies how PRSD adjusts to meet a much more complex product design with highly uncertain and uncontrollable security risks.

Finally, the fourth case study features “Vacubot”, a vacuuming robot that exclusively takes input from arbitrary other OCF devices within the same home network. This scenario primarily demonstrates how PRSD operates when applied to a device with relatively small security risks.

Most fundamentally, these case studies demonstrate that, at least in these examples, PRSD is useful for coming to security decisions. It helps developers consider threats and appropriate security responses comprehensively and systematically. In each case, all aspects of analysis in each process proved useful at some point along the way towards reaching security decisions. Thus, skipping any aspects of the processes in the framework would have entailed missing aspects of analysis relevant to the final security decisions.

Between case studies, differing situations and threat models justify different decisions between similar competing security options. The only common security option between the case studies is encryption, which is explicitly required by OCF. Thus, as hypothesized, as tradeoffs vary with respect to situations and threat models, different security options become more or less reasonable than each other. PRSD helps grapple with these tradeoffs. A less systematic framework increases the risk that developers miss a crucial element of analysis, and a more rigid framework would lead to suboptimal security designs in some or all of the case studies. Based on these observations, then, it appears that PRSD is appropriately flexible to the current needs of home IoT.

Architectural Synthesis can do a lot to resolve tradeoffs in general, before developers have to make the hard choices themselves in Tradeoff Resolution. In these case studies, the OCF architecture’s constraints often eliminated some security options, provided guidance for choosing between others, and significantly affected the tradeoffs between them (mainly by making some options cheaper by providing standard implementations.) While Architectural Synthesis fell far short of resolving all tradeoffs—and there will always be some situations where developers should have some freedom to choose between security options—it is clear that the OCF and other architectural standards can have an enormous impact on security design.

Notwithstanding the help that it did provide, the case studies reveal that the OCF specification has several crucial gaps. First, it does too little to help developers select, design, or configure credential management systems, or CMSs.[6] While providing a full specification to all these things would, for good reason, fall outside of the scope of the OCF specification, standardizing or guiding aspects of CMS interfaces would help considerably and fall within the purview of the specification. Second, the ACL management section is explicitly incomplete. Permissions are notoriously difficult (see Watson et al. [89]); this is not an area that developers should have to figure out without guidance. Third, there no explicit specification for bringing guest devices onto an OCF network. The core specification simply assumes away the problem, even though the OCF architecture is clearly intended to provide such functionality. Fourth, it would generally increase security—with negligible tradeoffs—and make analysis more clear and manageable if devices had a way to declare that trust in them should be limited. However, this is impossible in the OCF architecture. Finally, the OCF noticeably chooses not to include software updates within its scope. While the OCF is free to make this decision, updates are a critical aspect of security, and the OCF specification is well positioned to give developers guidance about how to deliver updates.

Separately, it is relatively straightforward to update existing analyses to account for new information. By having a methodical design and explicit links between dependencies, outputs, and processes, PRSD makes it easy to determine which aspects of analysis are affected by new information and to update them with minimal unnecessary effort.

Similarly, PRSD scales well to more complicated projects. The Hub in particular is a more complex product, with a greater potential attack surface and great uncertainty in the potential harms. Still, the situational and threat modeling processes lent themselves to straightforward decompositions. In general, more complex projects will require more time to analyze, but clustering competing security options and

[6]CMSs are a construct of the OCF specification [67]. They are never fully specified, either in structure or in interfaces, but they are generally services that generate, issue, and revoke credentials for other OCF devices.

their sub-options together should prevent analysis from becoming too complicated.

PRSD also works for simple devices that introduce only low-risk threats, such as the Vacubot vacuuming robot. It may take more effort than ad hoc analysis, but it provides a system through which to rationally justify rejecting security options that are more secure but sacrifice too much in their tradeoffs. This could be particularly helpful if the organization needs to justify their decisions in court. Absent legal considerations, PRSD could still positively affect the bottom line, producing a low-risk device by allowing the organization to see that it is reasonable to choose security options that have more attractive tradeoffs. Having more features and better usability ideally encourages greater demand from consumers, and lower security costs saves money. Thus, despite the higher costs than ad hoc analysis, there are several plausible ways this extra effort could provide a worthwhile return on investment.

4.3 Summary

The Processes for Reasonably Secure Design is a process-based security framework that helps developers of home IoT products compatible with the Open Connectivity Foundation communications architecture determine what security designs will be most reasonable for the situations their devices will face. To accomplish this, it walks developers through the process of considering potential threats, options for mitigating those threats, and the analysis of the tradeoffs between those options. Even though security design is a critical source of problems for the Internet of Things, no other framework provides this kind of guidance.

In addition to presenting this novel framework, we demonstrate its use with four illustrative case studies. In doing so, we observe that PRSD fulfills its intended goals with appropriate flexibility for the subjects within its scope, and it does so with minimal unnecessary analysis. Additionally, we find that PRSD holds up well for large and small projects, even as they grow close to the boundaries of its scope.

Separately, our enquiries revealed several notable gaps in the OCF specification. They lack any guidance or specification for credential management and ACL management services, and they do not specify how to interact with guest devices. Both of these are major shortcomings that fall at least some degree within the specification's scope. Additionally, it would be nice if the specification provided a way that allowed an endpoint to limit how much others trust it, but this concept is impossible within the OCF structure. Likewise, the choice to exclude software updates from the scope of the OCF architecture is suboptimal.

Chapter 5

Evaluation of PRSD

5.1 Quality of Guidance

The primary novel contribution of the Processes for Reasonably Secure Design is that its guidance aims for reasonable security that varies with stakeholder perspective, assets, and context. Furthermore, it provides system for explicitly identifying and comparing the tradeoffs between security options. Existing process-based security frameworks often prescribe rigid security goals, such as the CIA triad, or mechanisms, such as requiring certain kinds of authentication or encryption, irrespective of the tradeoffs that may arise from context. Even when frameworks do admit that certain security mechanisms may not be appropriate for all situations, such as the case with NIST 800–160 and the CII Best Practices Badge Program, they do not provide any guidance to help developers actually decide whether or not a security mechanism is worth its tradeoffs for the product they are making.

The hypothetical case studies in Appendix B concretely demonstrate how PRSD meets its design goals. It is actually possible to specify situational considerations, threat models, security options, and the tradeoffs between those options to a great enough extent that they are useful for making decisions about reasonable security. To be clear, it does not try to tell developers the "right" way to make security decisions—such decisions are value judgements that PRSD has no place imposing on developers. Rather, it simply helps developers identify and organize the information that is relevant to making these decisions.

As discussed in previous chapters, this focus on reasonable security and weighing security option tradeoffs is factually correct and, by definition, reasonable. Additionally, based on its design structure, the PRSD framework is also timely, because none of it is obsolete, complete, because it fully covers the subjects within its scope, and it is not misleading, because its documentation is clear that it does not cover all aspects of security. Thus, for what it tries to do, PRSD appears to have a high quality of guidance.

5.2 Flexibility

PRSD is designed to be highly flexible; the whole framework is designed to help developers determine what security options are right for their product given the circumstances they are operating in. The evaluation here, then, must determine both whether or not this flexibility goal is appropriate and whether or not it is met.

The case studies demonstrate that different decisions can be made among similar competing security options, depending on the stakeholder perspectives, assets, and contexts that developers expect their products to be used in. A more rigid framework would have led to suboptimal security designs in some or all of the case studies. From this, we can conclude that PRSD is not too rigid. At the same time, every output of every process at some point contributed to the final decisions, which were not always intuitive. As a result, a less comprehensive and methodical framework would make it more difficult to come to an appropriate security decision, so it is not too flexible. Thus, PRSD meets its goal and is appropriately flexible.

PRSD is specifically catered for home IoT implementing the OCF specification. For other verticals or aspects of design, it could be too flexible or too rigid. To generalize PRSD to other domains, its flexibility should be adjusted accordingly.

5.3 Administrative Costs

The administrative costs of using PRSD are probably low and manageable. These costs arise from three general categories: the useful work needed to actually perform the analyses, any unnecessary work required by the framework, and any overhead tasks that do not directly contribute to the analyses. The total administrative costs, then, are these costs minus the costs that a given organization would spend in those areas if they did not use PRSD.

Of these categories, the useful work required by PRSD will by far be the most costly. Using PRSD requires a fair amount of critical thinking, which is inherently difficult and resource-consuming. Since PRSD helps developers in part by leading them to consider situations, threats, security options, and tradeoffs that they would not think of otherwise, this work will usually be greater than what an organization would put towards security design otherwise.

The case studies contained very little unnecessary effort; all outputs tended to be useful at some point in the process leading to the security decisions. One might argue that too much time is spent considering some security options before realizing that they are not supported by the OCF, such as what occurred with password authentication options. Admittedly, this kind of analysis is probably unnecessary in hindsight, but this approach may be worth the cost.[1] Besides that, PRSD could actually help decrease unnecessary analysis over ad hoc analysis. Thus, although it can introduce a little unnecessary analysis, there are situations where PRSD could actually help developers avoid unnecessary analysis overall.

The only work that PRSD requires outside of analysis itself is documentation of the outputs of every process. While small, this documentation is not negligible.

[1]The alternative is to rely on the OCF specification as the arbiter of security options. While this might help developers avoid considering forbidden security options in the first place, it also risks serving as a crutch to coming up with security options. This outcome could lead developers to discount useful, acceptable options that are not explicitly written into the OCF specification. There is probably a more ideal compromise between the existing approach and the alternative presented here, but there is no way to tell what that ideal is until developers start actually using PRSD. Regardless, as developers become more familiar with the OCF specification, this corner case will come up less and less frequently.

As design details become more and more granular, and documentation is made for increasingly detailed sub-options to security decisions, this overhead will likely become particularly noticeable. However, because the framework itself does not specify any particular kind of formatting for the documentation, companies can produce whatever documentation suits their needs. For a startup designing a simple product, PRSD outputs could be scrawled out on a whiteboard. For a large multinational coordinating security designs and OCF implementations across several complex products, more elaborate and formal documentation might be in order. This flexibility should ideally ensure that the extra administrative costs of using this framework always remain manageable.

By examining the structure of PRSD itself and the case studies, we predict that PRSD entails low and manageable administrative costs. As with NIST 800–160 in Section 3.2, empirical evidence is necessary to determine how well these predictions hold in practice.

5.4 Scale

The case studies in Appendix B demonstrate that PRSD works for simple products with minimal OCF interactions, as is the case with the smart lock, as well as for more complex products with an uncertain range of possible OCF interactions, as is the case with the Hub. Based off of these examples, PRSD appears to scale well to small and large projects.

To be sure, the more complex projects did entail more complex case studies, but that is only because more complex projects are, by definition, more complex to design. As a design-focused security framework, there is no way that PRSD can get around this.[2]

[2]To scale poorly, the marginal amount of analysis that PRSD requires would have to increase with the complexity of a project. In other words, the work required by PRSD would have to grow exponentially with the complexity of a project. We predict that the work required by PRSD does not grow exponentially with complexity. The initial analysis of the Hub did require more effort than the other case studies, but there is nothing in it to suggest that the difference is exponential. When delving into sub-options throughout development, the difference in analytical workload between simple and complex projects is likely to grow. However, since more granular reapplications of PRSD

5.5 Scope

The PRSD framework is designed specifically for helping developers arrive at security designs for devices intended to be used primarily in homes that are compliant with the Open Connectivity Foundation architecture. More specifically, it is only relevant to how the product interfaces with and realizes the OCF specification and the threats enabled by that realization. Examination of the details of the framework in Appendix A clearly demonstrate that PRSD adheres to its scope throughout its structure.

5.6 Incentives

There are four potential future paths for PRSD adoption, and each one brings with it unique factors affecting the incentives that firms will have to use the framework. By default, PRSD will remain a standalone voluntary framework, in which firms designing OCF-compliant home IoT products can choose to use it based on its reputed quality, flexibility, and costs. Alternatively, some legislating body could enact laws that require firms to use PRSD when designing in-scope products. As a less strict alternative, the FTC and U.S. Courts could use PRSD as the basis for building a common law IoT liability framework. Here, the FTC would work with relevant stakeholders to build guidelines for designing home IoT products with reasonable security, and firms would be able to use these guidelines and PRSD as a defense against legal liability. Finally, the Open Connectivity Foundation itself could require organizations to use PRSD as part of getting their devices certified. In doing so, PRSD could become incorporated into the OCF standard. We delve into each of these paths to assess their plausibility and effects on the incentives to use PRSD.

only consider competing sub-options (instead of all sub-options at once), the analytical complexity should remain relatively stable and tractable throughout. While we cannot know for sure until PRSD is used on real products, this reasoning leads us to hypothesize that PRSD will tend to scale linearly, not exponentially, with complexity.

5.6.1 Voluntary Framework

In this situation, an organization's choice to use PRSD rests solely on their awareness of the framework and the perceived costs and benefits of using it. On its own, the administrative costs of using PRSD would disincentive its use. The IoT market is highly competitive, and vendors are racing to market for the first mover advantage [78, pp. 29–32]. Developers may eschew security design costs in an attempt to gain this advantage.

However, PRSD could provide a long term net benefit to organizations by helping them avoid costly security breaches or convincing them to forego security mechanisms that require great sacrifices in features and usability (thus lowering value to consumers and potentially decreasing demand) or cost. If a profit-driven organization became convinced that the expected benefits of PRSD exceed the short term costs, then they would have a net incentive to use it.

Whether or not PRSD ever gains a reputation of being worthwhile remains to be seen. In order to gain a reputation at all, the framework needs to be marketed in some way to convince organizations to try it. Beyond that, its reputation depends on word of mouth between CISOs and developers. However, it is unclear who would market and maintain the PRSD framework, so it is doubtful that PRSD would ever gain a widespread reputation as a voluntary framework.

It is worth noting that, in this situation, the developers using PRSD determine what constitutes "reasonable security" by themselves. When the organization is a profit-driven corporation, it's possible that some security threats will be ignored if they would not impact the corporation's bottom line. A FLOSS project, on the other hand, would have a different set of incentives. FLOSS developers want to gain a reputation for contributing to great projects. Thus, they want to build feature-rich and usable devices to impress their communities, but they also do not want to be responsible for a product that gains notoriety for insecurity [56].

5.6.2 Legislation

To make PRSD significantly more compelling to firms, legislators could pass a law requiring firms producing OCF-compliant home IoT devices to use a framework like PRSD throughout development. In this case, the incentives to use the framework would include the framework's reputation (as described in the previous section) and the expected legal penalties for not using PRSD, which would be the probability of an audit times the magnitude of a fine.[3] With a reasonably high risk of audit and sizable fines, this would be a compelling incentive.

Even ignoring the many difficulties in writing and enforcing such a law effectively,[4] there are no stakeholders with both the power and motivation to see this legislation through. Authority in this path ultimately lies with legislating bodies, whose motivations reflect those of representatives' constituents and donors. Companies might have the influence to lobby for these regulations successfully, but most companies would oppose these regulatory burdens.[5] Experts and advocacy groups may want to have a legal mandate on PRSD use, but they would probably have little clout to influence legislators on their own. Elected representatives would likely only take note of experts and advocates if their constituents started demanding it; however, constituents are unlikely to become aware of PRSD, because it is developer-oriented, so they are unlikely to call for regulatory action on it. As a result of these misalignments among stakeholders, PRSD is highly unlikely to become mandated through the formal legislation process.

5.6.3 Common Law Liability Framework

Alternatively, PRSD could become a mechanisms through which firms defend the reasonableness of the security designs of their products in court. In this situation, courts would settle upon standards for recognizing the use of PRSD when assessing

[3]There are also the indirect costs of the bad publicity that would accompany the fine.

[4]For instance, it would be difficult to update the legislation frequently enough to avoid holding back security innovations.

[5]Regulatory capture is not a concern, since PRSD scales well [83].

the security designs of products. Firms could then demonstrate the reasonableness of their security decisions through the outputs of their PRSD analyses. To catalyze this approach, the Federal Trade Commission could act as a focal point for bringing together relevant stakeholders and establish industry guidelines that would help companies use PRSD and help courts evaluate the use of PRSD properly.

There is an existing need for courts to set standards for sufficient information security. The FTC has already demonstrated the ability to bring charges against IoT companies for having fraudulent security in their case against TRENDnet [32]. While TRENDnet settled this case, the FTC did set the precedent of demonstrating that failure to achieve adequate security is an "unfair business practice" in its case against Wyndham Worldwide Corporation [87]. However, there are no clear standards defining exactly what adequate cybersecurity is. The courts will eventually establish this standard through precedents set by individual cases, and the FTC, with its newfound authority to initiate and win these cases, will likely play an active role in this process.

To do so, it is plausible that the FTC will facilitate the development of a common law framework for determining reasonable security similar to how it has for privacy.[6] In this scenario, the FTC would regularly hold workshops and forums to bring together industry, academia, experts, advocacy groups, and regulators to discuss reasonable approaches to IoT security design. PRSD could serve as useful independent analysis towards this end, although these discussions would generalize beyond the framework's scope. If court findings support the results of these discussions, the threat of an FTC complaint would incentivize the use of whatever variation of PRSD

[6]The FTC has built a framework of privacy standards that is effectively a body of common law, based on a couple regulations that "establish a floor" and "enforce the minimum" acceptable privacy standards. From this foundation, the FTC has used its power to take action against "unfair and deceptive practices" under Section 5 of the FTC Act to build a "patchwork" of regulations and norms that guide firms as they develop their privacy policies. To do so, the FTC brings together a variety of stakeholders, including industry, academia experts advocacy groups, and regulators, to determine any developments relevant to privacy, evaluate existing privacy practices, and ultimately agree upon changes to the common law framework. Although many aspects of this privacy framework are not actual law, companies still do generally abide by the guidelines, as doing so keeps the FTC from enacting more cumbersome compliance-based regulations and also preserves their seats in these workshops [80, 15].

the FTC supports.

5.6.4 Standardization

The Open Connectivity Foundation could provide a large incentive to use PRSD by requiring its use in home-based OCF-certified products.[7] More specifically, the OCF would take charge of maintaining and improving PRSD, and it would provide guidance to companies to make the framework easier to use. Certification testers would enforce proper use of the framework. Any companies wanting to produce OCF-compliant home devices would thus have to use PRSD in order to gain official OCF approval.[8]

If the OCF standard gains widespread adoption—making OCF compliance more valuable and thus more attractive—the benefits of OCF certification could easily trump the administrative costs of using PRSD. Furthermore, since OCF would require all developers of home products to use PRSD, firms would not have to worry about other OCF-compliant competitors skipping security design to get first mover advantage, diminishing the disincentive of the administrative costs in the first place.

It is unclear that the OCF would choose to require the use of PRSD. The OCF wants its standard and compliant products to attract market demand, but this requires IoT producers to support the OCF and standardize it in the first place. By requiring the use of PRSD, the Open Connectivity Foundation could gain publicity for supporting good security design. Additionally, with time, the positive effects of PRSD could make OCF compliance a strong market signal of reasonable security. However, OCF is not currently a widespread standard, and it does not want to discourage its adoption. While over 300 organizations have joined the OCF [69], the added costs from requiring the use of PRSD, which include its administrative costs and increased certification costs, could scare them away from ever making OCF

[7]For more information on OCF certification, see [67].

[8]More specifically, this approach has the OCF leveraging the network effects of its standard to incentivize the use of PRSD. Knieps provides a great introduction to the economics of standardization and network effects [51].

products.[9] In making its decision, then, the OCF would need to balance the potential long term benefits of requiring PRSD with the practical considerations relevant to becoming the de facto IoT communications in the first place.

5.7 Summary

The Processes for Reasonably Secure Design comprises a process-based security framework that aims to help developers achieve reasonable security designs in their products, understanding that the appropriateness of a security design varies with the context that a product is used in. Both in respecting this variance and providing tools to help developers identify and compare tradeoffs between security options, the guidance in this framework is novel.

This chapter evaluates PRSD according to the criteria identified in Chapter 3. It finds that PRSD has a high *quality of guidance*, is appropriately *flexible* to its scope, and will probably have low, manageable *administrative costs*. Its *scope* covers the security designs of home IoT products as they relate to threats enabled by the OCF standards, and we predict that it will *scale* well to all sizes of projects in this space. The *incentives* that organizations will have to use (or not use) this framework are highly uncertain; there are plausible ways that PRSD could come to widespread use, but there are at least as many ways it could not.

Table 5.1 summarizes the evaluation PRSD, comparing it to the three frameworks from Section 3.2.

[9]Several standardization mechanics affect these factors. Key to standardization is achieving a *critical mass*, which is the amount of participation in a standard necessary for the network to sustain itself [51, p. 108]. The biggest threat to reaching this mass is the *penguin effect*, in which producers would decide not to create OCF products out of fear that no others will follow, even if they believe that OCF has the superior standard for their products [25].

Table 5.1: Summary of evaluations of PRSD and other security frameworks.

Criteria	NIST 800–160	CITL	CII Best Practices	PRSD
Quality of guidance	mixed	high	medium-high	high
Flexibility	too little	appropriately none	too little	appropriately high
Administrative costs	uncertain	minimal required[a]	very little	manageable
Scale	high[b]	maximum	high	high
Scope	organizational structure of all systems security	traditional executables binaries[c]	all FLOSS projects	design of home IoT with OCF standards
Incentives	uncertain	uncertain for developers[d]	uncertain	uncertain

[a]Additionally, manageable for testers
[b]Biased towards large projects
[c]There are plans to include IoT in the future
[d]Additionally, strong for testers

Chapter 6

Conclusions

To maximize the value of the Internet of Things, developers will need to engineer devices that achieve sufficiently secure designs without compromising too much in terms of features, cost, and usability. To do so, they should remain mindful that, since threats vary depending on perspective, the assets at risk, and the environment a device is used in, the security mechanisms in their designs should vary with those factors, too. While this understanding applies in all domains of computer security, the Internet of Things will force developers to grapple with previously unexplored interactions between information and actuation, so the perspectives, assets, and environments these developers will need to consider will be particularly unexplored. To help overcome these difficulties, this document identifies gaps in the guidance available to developers and proposes a novel process-based security framework that begins to fill them.

Many organizations have proposed security frameworks to help developers produce more secure IoT devices. Of these, outcome-based security frameworks remain theoretically attractive but practically infeasible. Process-based security frameworks seem useful, but, before now, there did not exist criteria with which to evaluate and compare them.

This work introduces six criteria useful for evaluating and comparing these frameworks:

1. Quality of Guidance
2. Flexibility
3. Administrative Costs
4. Scope
5. Scale
6. Incentives

We used these criteria to evaluate multiple process-based security frameworks (of which, three in particular are highlighted within this work.) Even though the evaluations remain qualitative, they revealed shortcomings in the guidance available to developers. Of particular note, these frameworks often derive from inflexible conceptions of security, and they do not give room to vary security designs with the situations that a particular product might face. When developers do have room to choose between security options, they lack the tools they need to enumerate their options and organize the information relevant to choosing between them. With these shortcomings, developers are prone to missing threats entirely, inadequately mitigating them, or choosing security designs that sacrifice too much in terms of features, cost, usability, or other aspects of security. Widespread vulnerabilities in existing IoT devices confirm that developers do actually make these mistakes regularly, to the point that consumer confidence in IoT as a whole is at risk.

The Processes for Reasonably Secure Design is a first step towards addressing these shortcomings. This novel process-based security framework helps developers comprehensively and systematically identify and consider the security threats an IoT device may introduce to its surroundings, options for appropriately mitigating those threats, and the tradeoffs between those options. In effect, it gives developers the tools they need to organize and consider the information necessary to making reasonable security designs for their products. It is particularly tailored for the designs in its scope, which include home IoT devices and threats enabled by their use of the

Open Connectivity Foundation communication standards. In addition to describing the framework itself, we concretely demonstrated its use in multiple case studies, providing evidence of its value in the process.

We evaluated PRSD, finding that its *quality of guidance* is high, meaning that it should successfully help developers in the way it is meant to; it is appropriately *flexible* for the scope that it is tailored to, and its *administrative costs* will probably be low and manageable. To consider how widespread its use could be, one should keep in mind that its *scope* is limited to the security designs of home IoT products as they pertain to the OCF standards, but that it should *scale* well to projects of all sizes within this scope. Additionally, there are many possible futures for PRSD; there are some plausible ones in which companies will have strong *incentives* to use PRSD, but there are others where they will not. Thus, if developers are incentivized to use the framework, PRSD is well-equipped to help developers create products with more reasonable security designs.

6.1 Future work

There are many ways to expand upon this work. Perhaps most fundamentally, the six criteria for evaluating process-based security frameworks could be expanded. While the qualitative evaluations here proved useful, more rigorous, formal, and quantitative routines for performing the evaluations could lead to more standardized, objective, and complete results.

The PRSD framework itself could be improved in several ways. As home IoT and the OCF standard stabilize, information about common analytical patterns and relevant technical guidance could also be included in order to make the framework appropriately more rigid. Beyond this, further work could generalize the PRSD approach to other architectures beyond the OCF standards and beyond home-based Internet of Things products. Finally, if it ever becomes clear that there are structural shortcomings with the processes and their connections, the framework should be updated as needed.

Beyond guiding developers' security engineering efforts, elements of PRSD could ultimately form a tool that helps understand the relationship between architectural constraints and security designs. Across the case studies, the Architectural Synthesis process makes it clear that architectures play a tremendous role in defining what threats are possible, what security mechanisms could possibly be used to combat them, and which parties must take responsibility for implementing various security mechanisms. By going through congruent hypothetical case studies that differ in communications architecture, one might be able to employ PRSD to explore how different architectures lead to different security designs. Over enough case studies, this approach might even be useful for generalizing theories about the role of communications architectures in security.

Finally, if either the Federal Trade Commission or the Open Connectivity Foundation adopted the Processes for Reasonably Secure Design and leveraged their positions to encourage its use, the framework would likely have a significantly greater positive impact on the security designs of home-based IoT devices.

Bibliography

[1] Lillian Ablon et al. “Consumer Attitudes Toward Data Breach Notifications and Loss of Personal Information”. In: *15th Annual Workshop on the Economics of Information Security (WEIS)*. 2016. ISBN: 9780833093127. URL: http://weis2016.econinfosec.org/wp-content/uploads/sites/2/2016/05/WEIS%7B%5C_%7D2016%7B%5C_%7Dpaper%7B%5C_%7D11-2.pdf.

[2] Alessandro Acquisti and Jens Grossklags. *Privacy and rationality in individual decision making.* 2005. DOI: 10.1109/MSP.2005.22.

[3] Anne Adams and Martina Angela Sasse. “Users are not the enemy”. In: *Communications of the ACM* 42.12 (1999), pp. 40–46. ISSN: 00010782. DOI: 10.1145/322796.322806.

[4] Airbnb. *Airbnb Unveils Expansive Suite of Personalized Tools to Empower Hosts.* 2015. URL: https://www.airbnb.com/press/news/airbnb-unveils-expansive-suite-of-personalized-tools-to-empower-hosts (visited on 12/28/2016).

[5] Amazon. *Amazon Echo - Amazon Official Site - Alexa-Enabled.* 2017. URL: https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E (visited on 05/04/2017).

[6] Ross Anderson. “Why Information Security is Hard”. In: *Annual Computer Security Applications Conference* (2001). URL: www.cl.cam.ac.uk/%7B~%7Drja14/%7B%5C#%7DEcon.

[7] Ross J. Anderson. “Usability and Psychology”. In: *Security Engineering: A Guide to Building Dependable Distributed Systems*. 2nd. Wiley, 2008. Chap. 2. ISBN: 9780470068526. DOI: `10.1093/epirev/mxr031`. URL: `http://www.cl.cam.ac.uk/%7B~%7Drja14/Papers/SEv2-c02.pdf%20http://portal.acm.org/citation.cfm?id=1373319`.

[8] Apple. *Approach to Privacy - Apple*. 2017. URL: `https://www.apple.com/privacy/approach-to-privacy/` (visited on 05/04/2017).

[9] Apple. *HomeKit*. 2017. URL: `https://developer.apple.com/homekit/` (visited on 05/04/2017).

[10] Apple. *HomeKit*. 2017. URL: `https://developer.apple.com/reference/homekit` (visited on 05/04/2017).

[11] Jari Arkko. *The Interoperability of Things: IoT Semantic Interoperability (IOTSI) Workshop 2016*. 2016. URL: `https://www.ietf.org/blog/2016/03/the-interoperability-of-things-iot-semantic-interoperability-iotsi-workshop-2016/` (visited on 02/09/2017).

[12] Ashish Arora et al. *Measuring the risk-based value of IT security solutions*. 2004. DOI: `10.1109/MITP.2004.89`.

[13] Matt Asay. *Why 10 million developers are lining up for the Internet of Things*. 2016. URL: `http://www.techrepublic.com/article/why-10-million-developers-are-lining-up-for-the-internet-of-things/http://www.techrepublic.com/article/why-10-million-developers-are-lining-up-for-the-internet-of-things/` (visited on 01/19/2017).

[14] August. *August Home Delivers Secure Access for Airbnb*. 2016. URL: `http://august.com/2015/11/12/august-now-works-with-airbnb-to-deliver-secure-and-easy-home-access/` (visited on 12/28/2016).

[15] Kenneth a. KA Bamberger and DK Deidre K. Deirdre Mulligan. “Privacy on the Books and on the Ground”. In: *Stanford Law Review* 63.247 (2011), pp. 247–316. ISSN: 00389765. DOI: `10.2139/ssrn.2230369`. URL: `http://scholarship.`

law.berkeley.edu/facpubs%20https://a.next.westlaw.com/Document/I23cca655362c11e08b05fdf15589d8e8/View/FullText.html?navigationPath=/Foldering/v4/elanazeide/folders/e652a265ef474b48a6c404123729d5d7/search/hsjpEfUjAxSWwJOAC5dllh2Uh6CcSCdmsyUeTCzj2W%7B%5C%%7D7Ca5hLTEyMYi4w%7B%5C%%7D60wJoaOsfkskOqSI2kbTFuhf.

[16] Joseph Bonneau and Paul C Van Oorschot. “The Quest to Replace Passwords : A Framework for Comparative Evaluation of Web Authentication Schemes”. In: 2012 (2012). DOI: 10.1109/SP.2012.44.

[17] Joseph Bonneau and Sören Preibusch. “The Inconvenient Truth About Web Certificates”. In: *Economics of Information Security and Privacy III*. 1. 2010, pp. 121–167. ISBN: 9781441969668. DOI: 10.1007/978-1-4419-6967-5. URL: http://www.springerlink.com/index/10.1007/978-1-4419-6967-5.

[18] David D Clark and Marjory S Blumenthal. “The End-to-End Argument and Application Design : The Role of Trust”. In: *Federal Communications Law Journal* 63.2 (2011).

[19] *CoAP — Constrained Application Protocol*. 2017. URL: http://coap.technology/ (visited on 05/04/2017).

[20] Core Infrastructure Initiative. *CII Best Practices Badge Program*. 2017. URL: https://bestpractices.coreinfrastructure.org/.

[21] Core Infrastructure Initiative. *CII Best Practices Projects*. 2017. URL: https://bestpractices.coreinfrastructure.org/projects.

[22] Core Infrastructure Initiative. *Other criteria for higher-level badges*. 2017. URL: https://github.com/linuxfoundation/cii-best-practices-badge/blob/master/doc/other.md (visited on 05/06/2017).

[23] Paul A. David. “Clio and the Economy of QWERTY”. In: *The American Economic Review* 75.2 (1985), pp. 332–337. ISSN: 00028282. DOI: 10.2104/ha080079. arXiv: /www.jstor.org/stable/1805621 [http:].

[24] Elise. *Data security & privacy on Google Home*. 2017. URL: https://support.google.com/googlehome/answer/7072285?hl=en (visited on 02/09/2017).

[25] Joseph Farrell and Garth Saloner. “Competition, Compatibility and Standards: The Economics of Horsese, Penguins, and Lemmings”. In: *Department of Economics, UCB* (1987).

[26] Federal Trade Commission. “IoT Privacy & Security in a Connected World”. In: (2015), p. 71.

[27] Roy Thomas Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. In: *Building* 54 (2000), p. 162. ISSN: 1098-6596. DOI: 10.1.1.91.2433. arXiv: arXiv:1011.1669v3. URL: http://www.ics.uci.edu/%7B~%7Dfielding/pubs/dissertation/top.htm.

[28] Fitbit. *Fitbit App*. 2016. URL: https://www.fitbit.com/app (visited on 05/07/2016).

[29] Sarah Flicker. “Stakeholder Analysis”. In: *The SAGE Encyclopedia of Action Research*. 2014. ISBN: 9781849200271. DOI: 10.4135/9781446294406.n319. URL: http://dx.doi.org/10.4135/9781446294406.n319.

[30] Dinei Florencio, Cormac Herley, and Van Oorschot Paul C. “Password Portfolios and the Finite-Effort User : Sustainably Managing Large Numbers of Accounts”. In: *Usenix Security*. 2014. ISBN: 9781931971157.

[31] Lorenzo Franceschi-Bicchierai. *Amazon Quietly Removes Encryption Support from its Gadgets*. 2016. URL: https://motherboard.vice.com/en%7B%5C_%7Dus/article/amazon-removes-device-encryption-fire-os-kindle-phones-and-tablets (visited on 02/09/2017).

[32] FTC. *FTC Approves Final Order Settling Charges Against TRENDnet, Inc.* 2014. URL: https://www.ftc.gov/news-events/press-releases/2014/02/ftc-approves-final-order-settling-charges-against-trendnet-inc (visited on 03/29/2017).

[33] Garmin. *Garmin Connect Mobile*. Garmin. 2016. URL: https://buy.garmin.com/en-US/US/on-the-go/apps/garmin-connect-mobile/prod125677.html (visited on 05/07/2016).

[34] Matthew Garret. *I've bought some more awful IoT stuff*. 2016. URL: http://mjg59.dreamwidth.org/43486.html (visited on 12/08/2016).

[35] Google. *Actions on Google*. 2017. URL: https://developers.google.com/actions/ (visited on 05/04/2017).

[36] Google. *Google Home*. 2017. URL: https://madeby.google.com/home/ (visited on 05/04/2017).

[37] Google Cloud. *Google Infrastructure Security Design Overview*. Tech. rep. Google, 2017. URL: https://cloud.google.com/security/security-design/%7B%5C%%7D0Ahttps://cloud.google.com/security/security-design/resources/google%7B%5C_%7Dinfrastructure%7B%5C_%7Dwhitepaper%7B%5C_%7Dfa.pdf.

[38] L Gordon and M Loeb. "The economics of information security investment". In: *ACM Trans, Inf. Syst. Sec.* 54.4 (2002), pp. 438–457.

[39] Daniel Hein and Hossein Saiedian. "Secure Software Engineering: Learning from the Past to Address Future Challenges". In: *Information Security Journal A Global Perspective* 18.1 (2009), pp. 8–25. ISSN: 19393555. DOI: 10.1080/19393550802623206. URL: http://www.informaworld.com/openurl?genre=article%7B%5C&%7Ddoi=10.1080/19393550802623206%7B%5C&%7Dmagic=crossref.

[40] Chad Heitzenrater, Rainer Böhme, and Andrew Simpson. "The Days Before Zero Day: Investment Models for Secure Software Engineering". In: *The Economics of Information Security and Privacy* (2016).

[41] Cormac Herley. "So Long , And No Thanks for the Externalities : The Rational Rejection of Security Advice by Users". In: *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*. 2009, pp. 133–144. ISBN: 9781605588452.

[42] Cormac Herley. “Unfalsifiability of security claims”. In: *Proceedings of the National Academy of Sciences* I.1 (2016), p. 201517797. ISSN: 0027-8424. DOI: 10.1073/pnas.1517797113. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.1517797113.

[43] Kevin J Soo Hoo. “How Much Is Enough? A Risk-Management Approach to Computer Security”. In: (2000). URL: https://cisac.fsi.stanford.edu/sites/default/files/soohoo.pdf.

[44] Huawei. *Global Connectivity Index 2015*. 2015. URL: http://www.huawei.com/minisite/gci/en/huawei-global-connectivity-index-2015-whitepaper-en-0507.pdf (visited on 11/18/2015).

[45] Internet Society. “The internet of things: an overview”. In: October (2015). URL: http://www.internetsociety.org/doc/iot-overview.

[46] IoTivity. *Documentation*. 2016. URL: https://www.iotivity.org/documentation (visited on 12/09/2016).

[47] IoTivity. *Home | IoTivity*. 2016. URL: https://www.iotivity.org/ (visited on 12/09/2016).

[48] ISO. *Quality Management Systems—Fundamentals and Vocabulary*. Tech. rep. ISO 9000:2015. 2015.

[49] ISO. *Information technology—Security techniques—Information security management systems—Overview and vocabulary*. ISO 27000:2016(E). Geneva, Switzerland: International Organization for Standardization, 2016.

[50] Jacob Kastrenakes. *Amazon’s Alexa is everywhere at CES 2017*. 2016. URL: http://www.theverge.com/ces/2017/1/4/14169550/amazon-alexa-so-many-things-at-ces-2017 (visited on 01/10/2017).

[51] Günter Knieps. *Network Economics*. Springer, 2014, pp. 1–184. ISBN: 0521800951.

[52] Brian Krebs. *Who is Anna-Senpai, the Mirai Worm Author?* 2017. URL: https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/ (visited on 01/19/2017).

[53] Juhee Kwon and M. Eric Johnson. “The Market Effect of Healthcare Security: Do Patients Care About Data Breaches”. In: *14th Annual Workshop on the Economics of Information Security (WEIS)*. 2015.

[54] Kevin LaCroix. *Target Directors and Officers Hit with Derivative Suits Based on Data Breach*. 2014. URL: `http://www.dandodiary.com/2014/02/articles/cyber-liability/target-directors-and-officers-hit-with-derivative-suits-based-on-data-breach/` (visited on 04/17/2017).

[55] Gwanhoo Lee and Weidong Xia. “Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility”. In: *MIS Quarterly* 34.1 (2010), pp. 87–114. ISSN: 0276-7783.

[56] Josh Lerner and Jean Tirole. “Some Simple Economics of Open Source”. In: *The Journal of Industrial Economics* 50.2 (2003), pp. 197–234. ISSN: 00221821. DOI: `10.1111/1467-6451.00174`. URL: `http://doi.wiley.com/10.1111/1467-6451.00174`.

[57] Level 3. *How the Grinch Stole IoT*. 2016. URL: `http://www.netformation.com/level-3-pov/how-the-grinch-stole-iot` (visited on 05/04/2017).

[58] James Manyika et al. *the Internet of Things : Mapping the Value Beyond the Hype*. Tech. rep. June. 2015.

[59] Charlie Miller and Chris Valasek. “Remote Exploitation of an Unaltered Passenger Vehicle”. In: *Blackhat USA* 2015 (2015), pp. 1–91. URL: `http://illmatics.com/Remote%20Car%20Hacking.pdf`.

[60] Tyler Moore and Ross Anderson. “Economics and Internet Security : a Survey of Recent Analytical , Empirical and Behavioral Research”. In: *Peitz, M., Waldfogel, J. (Eds.), The Oxford Handbook of the Digital Economy, Oxford University Press*. 2011.

[61] Tyler Moore, Scott Dynes, and Frederick R Chang. “Identifying How Firms Manage Cybersecurity Investment”. In: *Workshop on the Economics of Information Security (WEIS)* (2016).

[62] Nest Labs. *Nest*. 2017. URL: `https://nest.com/` (visited on 05/04/2017).

[63] NIST. “Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems”. In: (2016).

[64] Onstar. *RemoteLink Mobile App | OnStar*. 2016. URL: `https://www.onstar.com/us/en/services/remotelink.html?source=ct` (visited on 05/07/2016).

[65] Open Connectivity Foundation. *Certified Products Archive – Open Connectivity Foundation (OCF)*. 2016. URL: `https://openconnectivity.org/certified-products` (visited on 01/11/2017).

[66] Open Connectivity Foundation. *Oic Resource Type Candidate Specification v1.1.0*. Tech. rep. 2016. URL: `https://openconnectivity.org/resources/specifications`.

[67] Open Connectivity Foundation. *Oic Security Candidate Specification v1.1.0*. Tech. rep. 2016. URL: `https://openconnectivity.org/resources/specifications`.

[68] Open Connectivity Foundation. *Oic Smart Home Device Candidate Specification v1.1.0*. Tech. rep. 2016. URL: `https://openconnectivity.org/resources/specifications`.

[69] Open Connectivity Foundation. *OCF - About*. 2017. URL: `https://openconnectivity.org/about` (visited on 05/04/2017).

[70] Stephen Papa, William Casper, and Tyler Moore. “Securing wastewater facilities from accidental and intentional harm: A cost-benefit analysis”. In: *International Journal of Critical Infrastructure Protection* 6.2 (2013), pp. 96–106. ISSN: 18745482. DOI: `10.1016/j.ijcip.2013.05.002`.

[71] Arun Ross, Jidnya Shah, and Anil K. Jain. “From template to image: Reconstructing fingerprints from minutiae points”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.4 (2007), pp. 544–560. ISSN: 01628828. DOI: `10.1109/TPAMI.2007.1018`.

[72] Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems A . Considerations Surrounding the Study of Protection". In: *Proceedings of the IEEE* 63.9 (1975), pp. 1278–1308. ISSN: 00189219. DOI: 10.1109/PROC.1975.9939. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.357.2145%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf%20http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=1451869.

[73] Runa A Sandvik and Michael Auger. *When IoT Attacks: Hacking A Linux-Powered Rifle*. [YouTube video]. 2015. URL: https://www.youtube.com/watch?v=ZcukSF7ruZY%7B%5C&%7Dfeature=youtu.be (visited on 01/25/2017).

[74] M Angela Sasse et al. "Debunking Security – Usability Tradeoff Myths". In: *IEEE Security and Privacy* 14.5 (2016).

[75] Kammi Schmeer. *Guidelines for Conducting a Stakeholder Analysis*. Bethesda, MD: Partnerships for Health Reform, Abt Associates Inc, 1999, p. 42. URL: www.phrproject.com..

[76] Roy Schmidt et al. "Identifying Software Project Risks: An International Delphi Study". In: *Journal of Management Information Systems* 17.4 (2001), pp. 5–36.

[77] Bruce Schneier. *Security and Privacy Guidelines for the Internet of Things*. 2017. URL: https://www.schneier.com/blog/archives/2017/02/security%7B%5C_%7Dand%7B%5C_%7Dpr.html (visited on 05/05/2017).

[78] Carl Shapiro and Hal R Varian. *Information rules: a strategic guide to the network economy*. Harvard Business Press, 2013.

[79] Timothy Simcoe. "Standard setting committees: Consensus governance for shared technology platforms". In: *American Economic Review* 102.1 (2012), pp. 305–336. ISSN: 00028282. DOI: 10.1257/aer.102.1.305.

[80] Daniel J Solove and Woodrow Hartzog. "The FTC and the new common law of privacy". In: *Colum. L. Rev.* 114 (2014), p. 583.

[81] Sonos. *Music Control App*. Sonos. 2016. URL: `http://www.sonos.com/controller-app` (visited on 05/07/2016).

[82] Michael Spence. "JOB MARKET SIGNALING." In: *Quarterly Journal of Economics* 87.3 (1973), pp. 355–374. ISSN: 00335533. URL: `http://libproxy.mit.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bah&AN=4966646&site=ehost-live&scope=site`.

[83] George J Stigler. "The theory of economic regulation". In: *The Bell journal of economics and management science* (1971), pp. 3–21.

[84] Tesla. *Android and iPhone app*. 2016. URL: `https://www.teslamotors.com/support/android-and-iphone-app` (visited on 05/07/2016).

[85] Troy Hunt. *CloudFlare, SSL and unhealthy security absolutism*. 2016. URL: `https://www.troyhunt.com/cloudflare-ssl-and-unhealthy-security-absolutism/` (visited on 05/06/2017).

[86] Ultimate Ears. *UE APPS*. 2016. URL: `http://origin.ultimateears.com/ue-apps/en-us/` (visited on 05/07/2016).

[87] United States Court of Appeals for the Third Circuit. *Federal Trade Commission v. Wyndham Worldside Corporation*. 2015. URL: `http://www2.ca3.uscourts.gov/opinarch/143514p.pdf`.

[88] U.S. Department of Homeland Security. *Strategic Principles for Securing the Internet of Things (IoT)*. Tech. rep. U.S. Department of Homeland Security, 2016, pp. 1–17.

[89] Robert N M Watson et al. "Capsicum: practical capabilities for UNIX". In: *19th Usenix Security Symposium* (2010), p. 3. ISSN: 03601315. DOI: `10.1145/2093548.2093572`. URL: `http://dl.acm.org/citation.cfm?id=1929820.1929824`.

[90] Nicky Woolf. *DDoS attack that disrupted internet was largest of its kind in history, experts say*. 2016. URL: `https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet` (visited on 01/25/2017).

[91] Peiter Zatko and Sarah Zatko. *Project CITL*. [YouTube video]. 2016. URL: https://youtu.be/GhO9vyW1f7w?list=PL7gTU7vlLWKN-ca2ha0cYJBpR_pQKvMfa (visited on 05/05/2017).

[92] Igal Zeifman, Dima Bekerman, and Ben Herzberg. *Breaking Down Mirai: An IoT DDoS Botnet Analysis*. 2016. URL: https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html (visited on 10/25/2016).

[93] Zigbee Alliance. *Zigbee*. 2017. URL: http://www.zigbee.org/ (visited on 05/04/2017).

Appendix A

Processes For Reasonably Secure Design

A.1 Introduction

In order to deal with the high uncertainty and complexity of Internet of Things security, developers need assistance weighing their options when creating their security designs. Few process-based security frameworks provide room to vary security designs based on the expected stakeholder perspectives, assets, and environments that a product could be used in. Of those that do, we are aware of none that provide the tools that developers need to determine what security designs would be most reasonable for the situations their products will face.[1]

This document proposes the Processes for Reasonably Secure Design (PRSD) process-based security framework to help fulfill this need for home IoT products[2] as they incorporate the Open Connectivity Foundation standards[3] into their designs. The purpose of the framework is to help developers comprehensively and system-

[1]See Chapter 2 for a discussion of reasonable security and Chapter 3 for a look at other process-based security frameworks.

[2]We focus on the home IoT space, as it is a quickly growing segment of the IoT that involves interactions between a wide variety of heterogeneous devices, so it is an important sector at high risk of housing security vulnerabilities.

[3]We focus on the OCF standards, as they are the open communications standard with the most widespread industry support. Chapter 2 contains a high-level introduction to the OCF standards. See the OCF website for the complete specification [67].

atically identify and consider the security threats a (prospective or existing) device may introduce, options for appropriately mitigating those threats, and the tradeoffs between those options. Without having developers relinquish their responsibility to use their own judgement to weigh tradeoffs and ultimately determine security designs, PRSD offers them a coherent approach to identifying and prioritizing the information pertinent to carrying out that responsibility.

In this way, PRSD is unique among IoT security guidance in that it helps developers determine reasonable security designs relative to the particulars of their product and its use cases. It is also designed to be flexible enough to adapt to the changing needs of developers using it while still providing sufficient structure to help avoid errors, and to entail as little overhead tasks as possible. As a result, applying this framework should help developers design their OCF-compliant home IoT products with more reasonable security.

PRSD operates on existing external information and result in *security decisions*, which are security mechanisms that the developers have decided to include in the design of a product. Understanding that development teams rarely have a complete and final vision for a product at its inception [55], all analyses should be performed with whatever level of detail existing information permits. As changes are introduced, be they designs with greater granularity, new features, or changes to potential threats, developers can simply update the existing analyses accordingly. The analyses in PRSD are explicitly itemized and linked in order to make it easy to only reconsider the analyses affected by a change.

More specifically, the *processes* that comprise PRSD are separate but linked analysis operations that apply to their *dependencies* and produce *outputs*. Dependencies are information that influence the analysis of a process, and outputs are the conclusions resulting from that analysis. This document explicitly describes the dependencies, outputs, and the details of executing each process. As needed, it references various resources that may aid developers as they carry out process. Justifications for each process are also included; developers could use them to make informed decisions about modifying processes as needed to better suit their projects.

Generally speaking, developers should proceed through these processes by carrying out the processes as their dependencies become ready. Since most dependencies are the outputs of other processes, completing one process allows another to be carried out. Thus, developers can proceed from one process to the next until they have settled upon security decisions.

The first process, *Situational Modeling* (Section A.2), depends only upon existing external knowledge. Carrying out this process results in outputs that model the situations that a prospective product could be used in, including the product's functions, environment, and the relevant stakeholders. With this model, as well as some general threat expertise, developers may perform *Threat Modeling* (Section A.3), which outputs relevant security threats to the prospective product. With this, design can proceed to the next process, *Security Option Enumeration* (Section A.4), which involves coming up with security options for combating the threats and explicitly laying out the tradeoffs between those options. Next, the *Architectural Synthesis* (Section A.5) process allows developers to see the effect of architecture proposed by the Open Connectivity Foundation on the security options and their tradeoffs. Finally, any competing security options that the architecture does not settle are considered in *Tradeoff Resolution* (Section A.6), which results in the security decisions. Continually affecting all of the other processes, the *Updates and Maintenance* (Section A.7) process guides developers through the process of incorporating new information, design changes, and increased design detail into the other processes.

PRSD is specifically tailored for security threats enabled by the OCF architecture. These threats primarily include any harm that exploit communications over the architecture, monitoring OCF communications, bringing a device onto a local OCF network, and issues related to identification and trust between OCF products and services. For complete details, see the specification itself [67]. For comparison, some design aspects that fall outside of the PRSD scope include hardware, sensor, internal processing, and mechanisms for software updates. While those things certainly still affect security, PRSD may not be well suited to them. Well-understood aspects of design deserve more rigid and specific treatment, and less-understood aspects of design require more flexibility.[4]

Likewise, PRSD is intended to apply to home IoT products, which may include communications between one home IoT object and an external entity like a cloud server.[5] By the same reasoning that PRSD only applies to specific aspects of design, other IoT domains, such as connected cars and medical devices, as well as traditional computers are out of scope.

A.2 Situational Modeling

A.2.1 Dependencies and Output

This process requires any existing information about what kind of product the manufacturers aim to produce, including the situations it is intended to be used in and the audience it is intended for.

This process produces sets of *capabilities*, *environmental models*, and *stakeholder models*.

A.2.2 Process Details

Determining the appropriate security design for a home IoT product must start by carefully outlining the behavior of the product, the environments that it will be used in, and the stakeholders who could influence it or be influenced by it. The benefits and costs of using various security options vary with these factors, so tradeoffs cannot be determined before these factors are identified. This process guides developers through this identification.

Start by detailing the *capabilities*, or functions, that the product will be able to carry out. What are some situations that it will be useful for? What will its limitations be? This analysis should cover the entire life of the product, from initial

[4]See Chapter 3 for more discussion of framework flexibility.

[5]We use the term "product" to generally refer to whatever product is being designed and produced—it could refer to a standalone device, a subcomponent of devices, or a group of devices that are designed together.

setup to daily operation and, if needed, disposal.

It's important to consider both outcome capabilities and support capabilities. Outcome capabilities are actions and behaviors that the product is designed to be able to carry out. Support capabilities are capabilities that indirectly facilitate the product's ability to carry out its outcome capabilities. For instance, with an automated vacuum cleaner, driving around and vacuuming will be some of its outcome capabilities, and connecting to a smartphone to receive commands might be one of its support capabilities. The distinction between these two types of capabilities is not critical in itself to this analysis; we merely highlight them to help developers compile a more comprehensive list of capabilities for their proposed home IoT product.

Second, build the *environmental models*. These models include relevant characteristics of the situations that the product is designed to be used in. While PRSD does specifically focus on home IoT, there are still a variety environmental factors that can vary between homes. Environmental models will help to determine what stakeholder assets the product could potentially compromise and what factors will affect the functioning of this product. Environmental models might include, among other things, whether the product will be used inside or outside or other kinds of devices that might rely on the product for proper use.

The final step of this process is to create the *stakeholder models*.[6] For each stakeholder, one must consider what assets might be affected by the product, expected amount of expertise, time, and resources (funds, power, compute) each can and is willing to contribute towards influencing the product's development or operation. The kinds stakeholders include the following:

1. Users, who use the product[7]

2. Admins, who configure the product

3. Developers, who design and produce the product

[6]Flicker provides an overview of the process of stakeholder analysis [29], and Schmeer has published guidelines that may help to perform this analysis [75].

[7]It's important to note that users will have very limited expertise and a poor appreciation for how their decisions affect their security [2].

4. Support, who provide services to keep the products functioning normally (often overlap with developers)

5. Third parties, who provide products that the product depends on or which inadvertently affect the product (e.g., presumed ISPs, library developers, presumed firewall providers, etc.)

6. The public, who may be affected if the product affects some public good

Adversaries are also stakeholders in a product, but they are considered separately in the next section.

A.3 Threat Modeling

A.3.1 Dependencies and Output

This process depends on existing information pertinent to potential harms and threats, as well as the *cabilities*, *environmental models*, and *stakeholder models* produced in the situation process.

This process produces sets of potential *harms*, *attackers*, and *means*.

A.3.2 Process Details

It is necessary to have a clear idea of the potential threats enabled by a product in order to determine how to mitigate those threats; this process guides developers as they identify and analyze potential threats. Carrying out these tasks explicitly and systematically helps to avoid overlooking or underestimating obscure threats, correctly estimate risks, and even discard potential attacks that no actor would actually have an interest in performing.

Begin by mapping out the potential *harms* that a potential product enables. At this stage, simply note how a product's capabilities allow it to perform actions or inactions that could damage specific stakeholder assets. Determining why or how someone would cause this harm comes later.

Harms tend to fall into three categories. First, there are harms caused directly by the product. To compile these harms, consider how the product's capabilities could adversely affect anything in any of the environmental models, paying particular attention to stakeholder assets. Second, there are harms to the product itself. To estimate these potential harms, estimate the direct costs of damage to the product itself, such as repair and replacement costs, as well as additional harms incurred by other system components that rely on the product in question.[8] For example, with an IoT thermostat, harms caused by damage to the thermostat include replacement costs as well as the damage done by poorly controlled home temperature. Finally, there are harms indirectly enabled by exploiting the product. These harms include, for instance, exploiting the product in question in order to gain unauthorized access to a more valuable component of the home IoT system. They also include incorporating a product into a botnet (such as Mirai) for a distributed denial of service attack [92]. Together, these three categories of harms comprise the list of harms that will be used throughout the rest of this analysis.

For each of these types of harms, it may be useful to consider the classic pillars of confidentiality (sensitive data leakage), integrity (incorrect but possible behavior), and availability (inability to provide service.) While keeping these pillars in mind could help developers avoid overlooking harms, they do not cover all potential harms. With the IoT in particular, new interactions like sensing and physical actuation may enable harms outside the domains of these traditional categories.

Next, enumerate the potential *attackers*, who are stakeholders with interests in causing a potential harm. Identify actors that have an interest in causing each of the harms in the list. Consider both legitimate stakeholders and outsiders who might want to interfere with the system. This analysis mirrors the previous stakeholder analysis: for each attacker, identify the harms of interest, the expected expertise of the attacker, and the resources the attacker can and would be willing to contribute towards causing the harm.

[8]Harms incurred by products reliant upon a damaged product could also be framed as direct harm that results from neglect. Either framing is fine; these categories are only provided to help developers be complete.

Now, determine the relevant *means* through which attackers could cause the potential harms. Means are the ways through which attackers could theoretically cause the IoT product to cause harm. At this time, imagine that attackers have unlimited access to the product; do not assume any security mechanisms.[9] The relevant means are means that fall under the scope of the OCF specification. For instance, sending commands, eavesdropping on communications, and tampering with certificates used in authentication all exploit the OCF architecture and are thus in scope. For contrast, physically attacking the hardware and tampering with the manufacturing process would be out of scope. Means at boundary of the OCF architecture are up to the developers' discretion.

If there are any harms that no attacker would want to carry out, or if there are any harms that lack any means within the scope of the OCF specification, then those harms can be discarded from this analysis.[10]

Together, the sets of harms, attackers, and means comprise the threat model for the product. For organizational clarity, it is important to link harms with the attackers that would be motivated to carry them out and the means that could cause them.

A.4 Security Option Enumeration

A.4.1 Dependencies and Output

This process depends on external information about potential security techniques, the sets of *capabilities*, *environmental models*, *stakeholder models*, *harms*, *attackers*, and *means*.

[9]Assuming that all interfaces are vulnerable to attack forces practitioners to explicitly consider how to defend each one. This approach is specifically motivated by the Jeep Cherokee [59], TrackingPoint rifle [73], and Mirai [92] attacks. In each of these cases, the developers did not consider the security of all interfaces from the ground up. Doing so would likely have made the vulnerabilities in both the car and rifle obvious and easily fixed. For Mirai, it would have forced developers to consider how to deal with vulnerable SSH and Telnet services. As we will see, the logic of the upcoming processes would have made it clear that it is simply easier not to run SSH and Telnet than to rely on keeping universal vendor passwords secret.

[10]Alternatively, they could optionally be retained for future reassessment.

This process produces *security options* and the *tradeoffs* between them.

A.4.2 Process Details

With the situational factors and potential threats modeled, developers can look into security mechanisms to defend against those threats. This examination should include both the act of identifying potential mechanisms to defend against each harm—the *security options*—and analysis of the tradeoffs between them. In carrying out these steps sequentially, mechanism identification can be performed with a bias towards inclusivity, to avoid rejecting potential options prematurely. Likewise, explicitly and systematically considering tradeoffs between options helps developers to overcome intuitive biases and choose the appropriate options later.

To begin, for each of the means in the threat model, make a list of security options to prevent or deter an attacker from exploiting the means in question to cause any of the harms that can arise from it. Annotate each option with an estimate of the resources necessary to overcome it and carry out the means,[11] as well as any miscellaneous notes about its ability to defend against an attack.[12] Developers should organize this list into clusters of mutually competing security options.

Often, there will be too many variations of any given security option for developers to feasibly list all of them. Instead of completely specifying each security option, developers should only include as much information in each security option to make an informed decision between it and competing security options later. Later, after a particular security option is selected as a security decision, developers can return to this process and compare the various sub-options. For instance, if a team is considering using authentication, initial security option enumeration might consider using user-set passwords, cryptographic keys generated locally by the product, cryptographic keys issued by a global PKI system, and fingerprints. Here, generalizing both cryp-

[11]Although estimating the exact amount of capital required to overcome a security option is often infeasible, developers should try to make qualitative estimates that facilitate comparisons between security options.

[12]For instance, one might note that, if relying on the Certificate Authority, one stolen certificate could leave all dependent products vulnerable [17].

tographic key options into a single option would not include enough information to make an informed decision, as passwords and fingerprints compare very differently to keys generated locally and keys issued from a global authority. At the same time, including much more detail, such as splitting the fingerprint option into multiple options based on particulars of the scanning algorithm, would be unnecessary at this time, as the choice of algorithm likely would not significantly change how fingerprints compare to the other options. However, if the fingerprint option is eventually chosen as a security decision, developers can return to this process later to compare the potential algorithms and other sub-options. For more information on considering the sub-options of a security decision, see Section A.7.

Even with this top-down, iterative approach to comparing security options, it is often infeasible to enumerate every possible option. It is up to developers to balance comprehensiveness with budget and schedule constraints.

After initial security option identification is complete, it's time to analyze the tradeoffs between competing security options. The tradeoffs are as follows:

1. *security vs. features*: how do the security options constrain the product's ability to carry out its various capabilities? For instance, certain kinds of encryption may make it difficult to perform machine learning on the data generated and stored with the product.

2. *security vs. security*: how do the defenses of mutually exclusive security options compare? For instance, different mutually exclusive authentication methods have various advantages and disadvantages in terms of security alone.

3. *security vs. cost*: what costs will each security option put on each stakeholder? These costs come in many forms, such as the operational costs for users and admins, the development costs, and the costs of supporting the product. For instance, using a certificate authority (CA) for authentication requires the manufacturer or some third party to put in the effort to built and maintain the CA.

4. *security vs. usability*: how do the security options constrain the ability to operate the product? This is a special subset of security vs. cost that merits its own

consideration. For instance, requiring a large and complex password for every command to the product can be burdensome to users. When considering this tradeoff, note that there are times when sacrificing usability for security mechanisms that seem technically more robust can actually decrease security overall if users behave in more insecure behavior to circumvent security mechanisms that they do not like.[13]

After analyzing tradeoffs, if some of the options are obviously infeasible, then remove them from the set of plausible security options.

A.5 Architectural Synthesis

A.5.1 Dependencies and Output

This process depends on external information about the OCF architecture, which includes the OCF specification and potentially IoTivity documentation, as well as an initial *security options* and their *tradeoffs*.

This process prunes the *security options* and their *tradeoffs*.

A.5.2 Process Details

After determining ways to potentially secure a product in Security Option Enumeration, one must consult the relevant specifications to determine how to integrate each option with the OCF architecture. In doing so, some security options will probably be rendered impossible. With others, the specification may have guidance for weighing the tradeoffs between them and other options. In some cases, the OCF standardizes certain options, which can make them much cheaper to develop than if the developers had to roll them on their own.

Additionally, the architecture might lead the team to consider security options (or sub-options) that were not considered before. If this is the case, they should be

[13]Several esteemed security researchers discuss false tradeoffs between usability and security in a 2016 publication of *IEEE Security and Privacy* [74].

added to the list of security options and incorporated into the tradeoffs by reapplying the Security Option Enumeration process detailed in Section A.4.

A.6 Tradeoff Resolution

A.6.1 Dependencies and Output

This process relies on the *harms*, *attackers*, *means*, *security options* and their *tradeoffs*.

This process produces *security decisions*.

A.6.2 Process Details

At this stage, developers are ready to decide which security options to select as security decisions. To do so, they must resolve the tradeoffs between competing options, weighing the advantages and disadvantages of each in order to determine which security decisions will result in the most reasonable security designs.

The most reasonable security design will include the options that optimize the balance between harms that are expected to be mitigated and the features, security, costs, and usability that must be sacrificed to achieve those options. The outputs of the previous processes all contribute towards making these judgements. A security option will mitigate an attack, which is a combination of means, attacker, and harm, when the resources needed to circumvent it and carry out the means exceed the resources that the attacker is willing to invest to cause a harm. Thus, a security option's effect on overall risk can be estimated by examining the attacks it mitigates. Developers can then compare the estimated risk reductions and sacrifices of competing security options and decide on the set of options that best balances these factors.

PRSD is unique among process-based security frameworks, because it has developers perform these comparisons methodically. The Internet of Things introduces many tradeoffs that are complex and not obvious, and the wide variety of situations that products can be used in, even within the home context, can have a substantial effect on the tradeoffs between security options. As a result, making these comparisons

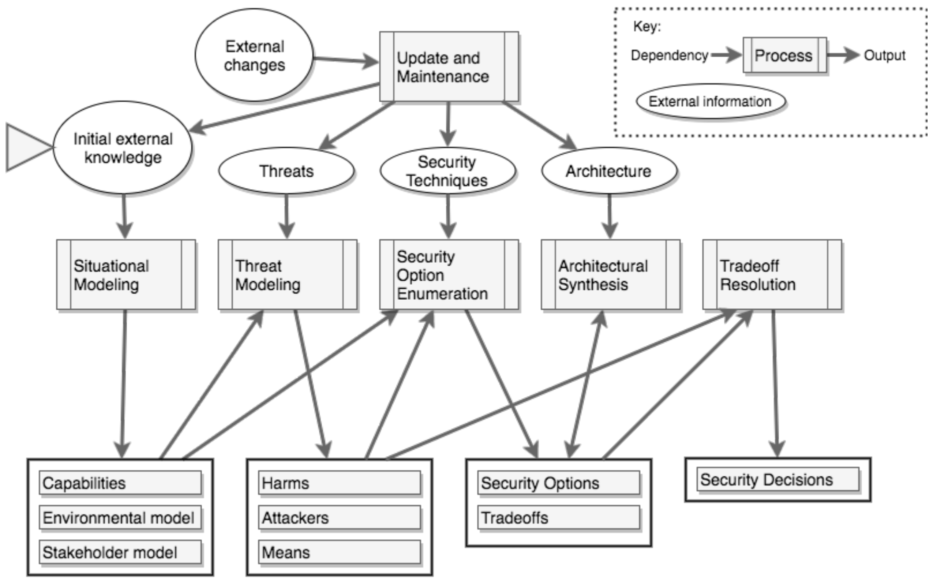

Figure A-1: Diagram of the Framework

with intuition or folk security advice alone is too superficial. PRSD helps overcome biases and simplistic advice, leading to products that are ultimately more reasonably secure.

A.7 Component Updates and Maintenance

A.7.1 Dependencies and Output

This meta-process depends on any external dynamics that may affect the product.

A.7.2 Process Details

To have project success, developers must account for rapidly changing circumstances, user needs, and design requirements [76], and modern software development methods like agile are designed to accommodate flexibility, incremental development, and changing requirements [55]. Furthermore, as current rampant IoT and software insecurity demonstrate, developers often do not fully secure their products, so they should anticipate needing to provide security updates during the lifecycle of their product. This meta-process describes how to incorporate changing circumstances into existing analysis under this framework.

When there is new information that might affect the product's security design, start by going through the dependencies of each process to see which processes are directly affected by the new information. New features, environments, and stakeholders should be incorporated into the Situational Modeling process, changing threats into the Threat Modeling process, etc. Increased details in the product's design should be incorporated the same way.

The modular structure of PRSD makes it relatively easy to propagate changes throughout relevant analyses without having to reconsider unaffected analyses. All process outputs should be linked to the information they depend on, so redoing a process is simply a matter of finding the outputs affected by the new information and limiting reanalysis to those dependencies and outputs. Likewise, whenever a process output is changed, the processes that use that output as a dependency can be updated in the same way. Figure A-1 illustrates the relationships between external information, processes, and their dependencies/outputs.

A.8 Maintaining the Framework

The Processes for Reasonably Secure Design can be adapted over time. As home IoT and the OCF standards stabilize, and security design patterns become more routine, more rigid resources will likely be developed in order to help developers avoid common errors. These more rigid resources will be compatible with PRSD, as they can be referenced and explained in the relevant processes. In this way, more rigid guidance would not replace PRSD so much as it would aid developers as they go through the processes.

This framework also anticipates that there may be shortcomings in the inherent structure of its processes. For this reason, the descriptions of each process justify the way they are structured. These justifications provide grounds for falsifying any part of its structure.[14] Unlike most process-based security frameworks, then, PRSD itself

[14]Falsifiability allows one to reject erroneous and unnecessary security practices, and it enables

provides a grounds from which to have a rational argument about its correctness and reasonableness.

one to prioritize accepted security practices [42].

Appendix B

Case Studies

This appendix contains four case studies that illustrate how to use the Processes for Reasonably Secure Design, the process-based security framework described in Appendix A. These case studies serve the dual purposes of providing concrete illustrations of the usage of PRSD and demonstrating certain aspects of the framework that support the claims made about its achievements. These case studies are limited to hypothetical situations, as there are no actual certified OCF products to which PRSD could currently be applied, even in hindsight.[1]

The first case study, in Section B.1, covers an OCF-certified door lock called "SuperLock." It exemplifies an IoT product that has simple behaviors but which requires significant security. This case study covers a high level initial application of PRSD, followed by a reapplication of PRSD to some sub-options of chosen security decisions.

The next case study, in Section B.2, starts off where the previous one finished, but with the company wanting to integrate with Airbnb services to automatically provide door access to Airbnb guests for the duration of their stays. This case study most notably demonstrates how to apply PRSD at the boundary of the OCF specification and third party proprietary interfaces, and it also exemplifies how to update existing analyses to accommodate changing design requirements.

[1]There are currently three OCF-certified products. [65], but none publish their design details, and they were not willing to discuss their designs with us.

Third, in Section B.3, we look at a “Hub”, which takes voice commands and responds with audio feedback to manage and operate an arbitrary variety of third party devices and cloud services. This case exemplifies how PRSD adjusts to meet a much more complex product design with highly uncertain and uncontrollable security risks.

Finally, Section B.4 covers the fourth case study, which features a vacuuming robot called “Vacubot” that exclusively takes input from arbitrary other OCF devices within the same home network. This scenario primarily demonstrates how PRSD operates when applied to a product with relatively small security risks.

B.1 Smart Door Lock

B.1.1 Given

Let’s suppose a startup called Lockr wants to make a smart OCF-compliant door lock marketed to Airbnb hosts, which they will call “SuperLock.” Often, it makes sense for hosts to give their guests house keys, but they might not want to pay the price to make many duplicates or have to worry about their guests losing or duplicating their keys. At the very least, it might give them more peace of mind if they can give guests expiring keys without having to pay for it. To make it as convenient as possible, the company wants to have smartphones communicate with SuperLocks to perform all configuration and operations. Lockr assumes that all smartphones are also OCF-compliant and capable of onboarding SuperLocks.[2] Lockr also assumes that smartphones can connect to the local OCF network, even remotely. A smartphone could thus either send requests to a SuperLock directly, if they are both on the local OCF network, or an intermediary device on the local OCF network could forward remote requests from a smartphone to a SuperLock, if the smartphone is not on the

[2]In the OCF specification, onboarding is the process of bringing a new device (in this case, a SuperLock) onto a local OCF network, making the network “own” it, and provisioning it without any necessary security constructs.

local OCF network.[3]

The company wants the locking mechanism to work as follows: The homeowner should have complete control over who can lock and unlock the door lock. She should be able to grant and revoke permission to use the lock conveniently and at will, and nobody without permission should be able to unlock the lock. The lock will not store logs or data aside from data strictly necessary for its ability to unlock and lock based on commands from trustworthy sources.

SuperLocks receive software updates over the air via a proprietary and explicitly non-OCF communications protocol. This is the only non-OCF communication it engages in.[4]

SuperLocks will have a backup physical lock mechanism, operated like a typical door lock with a physical key. Lockr has this so that people can still get into their homes if their OCF network goes down or if the SuperLock loses power, which is supplied by batteries. Additionally, SuperLocks' internals will be set up so that they will not lose their configuration when batteries are changed.

Lockr plans to use a tiny reset button on the indoors side of SuperLocks to allow homeowners to reset a SuperLock to factory settings.

Lockr wants SuperLocks to have mass appeal and be able to market the lock as more secure than a typical door lock. They are aiming to price it at around $50–100, even if that means sacrificing the market that requires exceptionally high security. In terms of its budget and schedule, Lockr is a pretty typical startup.

In this case study, we assume that the OCF architecture is still in the early stages of the standardization process. Most people have OCF compliant phones and local OCF networks in their homes with maybe a couple OCF devices. However, most devices are not OCF compliant, and no data types or design patterns beyond the minimum required by the OCF specification have become normalized.

[3]These are both scenarios that are specified and supported by the OCF specification, as explained in Section 5 of the Core specification v1.1.0 [67].

[4]The OCF specification inexplicably does not discuss software updates at all, so Lockr has decided to develop their own completely proprietary system for sending software updates to SuperLocks.

B.1.2 Situational Modeling

Capabilities

From the given information, we can specify several capabilities that SuperLocks will have. These are described in Table B.1.

Table B.1: SuperLock capabilities

Capability	Description
lock	lock and unlock mechanism
communication	Communication over the home's OCF network. These communications are intended to include both lock requests and administration with phones.
update	receive Internet updates over a proprietary protocol
physical lock	backup physical lock and unlock mechanism
reset	reset button restores SuperLock to factory settings

Environmental models

SuperLocks should serve the purposes of a wide variety of homes. The belongings in the home, which the lock should protect, will be of the utmost value to the homeowners. This lock should serve the purposes of homes that are in dangerous areas with high crime. We should assume that the house may be a valid target for somewhat experienced burglars. The locks will be accessible from both sides of the door, with the outside physically accessible to anyone who comes up to the door.

Stakeholder Models

The stakeholders are summarized in Table B.2. The table includes, for each type of stakeholder, the people who comprise that type of stakeholder, their interests and assets affected by SuperLocks, the amount of computer or security expertise they are assumed to have, and the amount of resources (e.g. time, effort, energy) they are willing to put towards SuperLocks.

Several aspects of these stakeholders merit elaboration. First, the "OCF network" asset reflects the local OCF network that a given SuperLock is connected to, including the other devices on that network. Additionally, the "availability" asset reflects whether or not the users and admins can use the SuperLock to get into the house.

Table B.2: SuperLock stakeholder Models

Type	Stakeholders	Interests and Assets	Expertise	Resources
Users	homeowners, Airbnb guests	SuperLock, safety, privacy, valuables, OCF network, availability	minimal	minimal
Admins	homeowners	SuperLock, safety, privacy, valuables, OCF network, availability	minimal	low
Developers	Lockr engineers	company revenue	high[a]	full time job
Support	Lockr engineers	company revenue	high[a]	full time job
Third parties	(none)			
Public	People who use the Internet	Internet access, time		

[a]Above average security expertise, compared to other developers.

The admins are in fact a subset of the users. They are the homeowners who put in a little extra time—reflected by the relatively greater "Resources" entry—to administer and configure the SuperLocks for their respective homes.

The developers and support are the same people, because Lockr is a startup that cannot afford a separate support team at this point in its life. We reason that Lockr has hired engineers with above average security expertise, because SuperLocks are designed specifically for security, so Lockr's reputation rests highly on delivering a secure product.

The public could be affected by SuperLocks if they are incorporated into a botnet. As part of a botnet, SuperLocks could either carry out DDoS attacks, which would hinder Internet access, or send spam, which would waste the public's time. We will examine this more in the Threat Modeling section.

B.1.3 Threat Modeling

Harms

There are several potential harms which could result from these door locks. The harms caused directly by the lock's capabilities derive from the lock letting unauthorized people into (and out of) the home and vice versa for homeowners and guests. In the former situation, people staying in the home could be attacked, the home could be vandalized, and things in it could be robbed. Additionally, just breaking in violates the sanctity of the home and is a harm in itself. In the latter, people could be kept out of the home, which, depending on the duration and situation outside the home (crime and weather), can range to highly inconvenient to, in extreme conditions, life threatening.

Additionally, the lock itself could be damaged. Besides leading to any of the aforementioned harms (depending on whether the lock is broken shut or open), this situation has the additional harm of repairing or replacing the lock. This would be a nonnegligible inconvenience in terms of both time and money for homeowners.

The lock could also be used as a pivot to exploit other devices on the local OCF network. For instance, attackers exploiting the lock might be able to get sensitive information from devices connected to the lock by masquerading as the lock.

Similarly, the lock could be turned into a bot in a botnet, which could be used to send spam email or contribute to Mirai-like distributed denial of service (DDoS) attacks.[5]

Table B.3 summarizes these harms.

Attackers

Table B.4 lists potential actors who may want to cause the harms made possible by SuperLocks.

For the most part, these attackers will have little computer security expertise, and they probably are not willing to put significant resources towards causing the harms. Exceptions include **government**, an attacker which has incredible expertise and resources, but would only target the SuperLocks of a few suspects or activists. **Hacktivists** can have a moderate amount of expertise,[6] but few resources to devote

[5]Although the only kind of Internet traffic that a SuperLock can receive is OCF requests from a smartphone, forwarded by a local intermediary, a hacked SuperLock could presumably send out any network traffic.

[6]We assume that hacktivist groups often have a few talented hackers and many script kiddies.

Table B.3: Potential harms resulting from SuperLocks

Harm	Description	Assets at risk
break in	attackers get into the house	safety, privacy, valuables, company revenue
locked out	users prevented from getting into home	safety, availability, company revenue
damaged	the lock is damaged	safety, privacy, valuables, company revenue
pivot harm	the SuperLock is exploited and used as a proxy to devices on the OCF network	OCF network, company revenue
bot	the SuperLock is incorporated into a botnet	Internet access, time, company revenue

Table B.4: Potential attackers with an interest in SuperLock harms. Exclamation points after harms indicate relatively higher expected interest in causing that harm, which in turn indicates a greater willingness to put resources towards causing that harm.

Attacker	Description	Harms of interest
burglars	criminals wanting to commit burglary	**break in** (!!), **damaged**
prankster	people looking to cause mischief	**break in**, **locked out**, **damaged**, **pivot harm**
government	agents of law enforcement or espionage	**break in**[a], **damaged**, **pivot harm**[a]
quarreling[b]	users fighting with each other	**locked out**
seizure[b]	landlords or other authorities seizing property	**locked out**
hacktivists	activists who make statements through hacking	**pivot harm** (!), **bot** (!!)
expansive crime	organized crime looking to steal data or amass botnets	**pivot harm** (!), **bot** (!!)

[a]States will have an strong interest in causing this harm to certain suspects or activists, but these cases are extremely rare, and those users will likely know the risks.

[b]It's possible that more harm could result from trying to protect against this attacker.

specifically to SuperLocks. Likewise, **expansive crime** groups will often have significant expertise and resources, but, since they focus on hitting as many targets as possible, they have only a few resources to devote to any individual target.

Stopping either **quarreling** or **seizure** from causing **locked out** could easily cause more harm than **locked out** would actually cause. Thus, we will not weigh in on this issue and discard these attackers from subsequent analysis.[7]

Means

The means includes any way to cause any of the harms through the OCF architecture. They are summarized in Table B.5.

Table B.5: Means through which attackers could cause harms by leveraging SuperLock OCF communications.

Means	Description	Harms
spoof admin	impersonating an administrator	**break in**, **locked out**, **damaged**, **pivot harm**, **bot**
admin commands	sending commands that entail administrator-level actions	**break in**, **locked out**, **damaged**, **pivot harm**, **bot**
spoof user	impersonating an authorized user	**break in**, **locked out**
user commands	sending commands that entail user-level actions	**break in**, **locked out**
MITM onboarding	man-in-the-middling the initial onboarding process	**break in**, **locked out**, **damaged**, **pivot harm**, **bot**

To elaborate, the "spoof" means involve pretending to be a legitimate admin or user, whereas the "commands" means involve sending commands that would get a SuperLock to take administrator or user-level actions without having to impersonate a legitimate admin or user. In other words, **spoof admin** and **spoof user** betray some failure of identification and authentication, and **admin commands** and **user commands** betray some failure of access control. Since we do not assume any particular security mechanisms at the threat modeling stage, we cannot go into more

[7]If Lockr did want to take a stance on this, they could, and the rest of the PRSD analysis could be adjusted accordingly.

detail about what exactly it would entail to carry out these means.

Once an attacker can successfully make administrator-level requests, we assume that the attacker essentially has root on the SuperLock. This assumption is reasonable, because administrator-level access would allow an attacker to take any action on the SuperLock's OCF resources, which essentially gives the attacker full read/write access to everything on the lock.[8]

B.1.4 Security Option Enumeration

Security options

It is infeasible to devise a comprehensive list of potential security options. Generally, there are often many plausible approaches to deal with each means, and there are often innumerable variations of each approach. Developers should consider as many options as possible, although they should understand that they will have to cut off their list of security options before it is complete. Here, we present a list of high-level security mechanisms that a company like Lockr might actually use.

encryption: use a strong encryption scheme to make all OCF communications confidential. While this doesn't prevent any means in itself, almost all of the other security options rely on this.

Next, we consider different approaches to identification and authentication, whose names will start with **auth**. These all rely on **encryption** to prevent attackers from stealing credentials. Additionally, for the time being, we will assume that properly implemented access controls accompany each of these options.[9] Authentication itself addresses **spoof admin** and **spoof user**, and the permissions system that accompanies it will mitigate **admin commands** and **user commands**.

[8]Under the OCF specification, an endpoint represents anything that it wants to expose to other OCF endpoints as "resources" that can be retrieved with RESTful requests. Furthermore, the OCF specification supports five generic operations: CREATE, RETRIEVE, UPDATE, DELETE, and NOTIFY, which each do what one would assume they do. With these operations, an administrator could make any conceivable change to a device's resources, which would almost surely allow her to get root.

[9]As Section B.1.7 demonstrates, developers can weigh options for access control details later, after deciding on a higher level authentication scheme.

1. **auth/password/vendor**: use a single manufacturer-set password listed in each SuperLock's instruction manual to authenticate each OCF request to a SuperLock, including during the onboarding process. This could slightly impede all of the means, but any attacker with access to SuperLock instructions could learn the password to any SuperLock.[10]

2. **auth/password/user**: have users set their own passwords for administration, permanent users (homeowners), and guests. This could stop most **burglars** and **pranksters**, but probably not **government**, **hacktivists**, or **expansive crime**.[11]

3. **auth/crypto/simple**: each SuperLock and smartphone generates their own public and private keys for authentication. This method would be extremely difficult to defeat, as it would require either stealing the credential, compromising a smartphone, or breaking public key cryptography—methods typically only available to **government**.

4. **auth/crypto/local**: have homeowners maintain a CMS[12] in their own homes to manage credentials for SuperLocks, smartphones, and other OCF devices. In addition to the potential weaknesses of **auth/crypto/simple**, an attacker could potentially steal an existing credential from a CMS or trick it into issuing her a new and privileged credential. Although Lockr cannot be sure of the structure of these local CMSs, only **government** would likely ever have sufficient means and interest to compromise a well designed local CMS to attack a SuperLock.[13] This would represent a single point of failure for all products that

[10]This might sound like a ridiculous option—it is. However, this was the approach that all the victims of Mirai took [92]. We include it here to demonstrate how PRSD reveals its weaknesses.

[11]Passwords can be guessed or socially engineered. Many people do not choose hard-to-guess passwords, even with most complexity rules [3]. Indeed, researchers have demonstrated that choosing complex and unique passwords for many accounts is not feasible in general [30].

[12]CMSs are a construct of the OCF specification [67]. They are never fully specified, either in structure or in interfaces, but they are generally services that generate, issue, and revoke credentials for other OCF devices. In this case study, we assume that CMS structure is no more clear to Lockr than it is in the real world now.

[13]This attack would not scale well enough for **expansive crime**'s interests, which is the only other attacker that might have the resources to attack a CMS.

rely on the CMS.

5. **auth/crypto/global**: use a single CMS to generate, issue, and revoke the credentials for all SuperLocks and smartphones.[14] This has the same types of weaknesses as **auth/crypto/local**, although the greater complexity of the global CMS both provides more attack vectors to attackers and also raises the potential harm if the CMS is compromised. As a result, in addition to **government**, **expansive crime** might be able and willing to circumvent this security option.

6. **auth/biometric**: have smartphones takes users' fingerprints and sends some representation of them to SuperLocks for authentication. To circumvent this, one would have to lift a user's fingerprint or break the underlying implementation, which could probably only be accomplished by **government** or perhaps **expansive crime**.

Additionally, there are several strategies to mitigate **MITM onboarding** and protect the initialization of a SuperLock, which we categorize with names that start with **ini**.

1. **ini/first**: on initial bootup, trust the first smartphone to contact the SuperLock to be the administrator for onboarding, and set up a secure channel to continue onboarding from there. This would prevent just about any attacker from using **MITM onboarding**, unless the attacker has a persistent presence on the local OCF network and is able to make contact with the lock before the legitimate administrator. Probably only **government** could do this consistently.

[14]There are many systems issues that Lockr would need to sort out to implement either CMS option. If the CMS under consideration already existed, then Lockr could analyze it with respect to SuperLocks accordingly. If Lockr had to make its own CMS, then that would require its own separate analysis. The Hub case study in Section B.3 represents an analysis of a device that acts similarly to a local CMS. A global CMS would fall outside the scope of PRSD. Instead of assuming a particular CMS, this case study treats the CMSs in these options as generic OCF CMSs. In doing so, we illustrate the shortcomings and difficulties in trying to reason about the vague concept described in the OCF specification.

2. **ini/PIN**: on initial bootup, display an out-of-band random PIN for the administrator's smartphone to enter in order to authenticate the administrator.[15] This would improve over **ini/first** by requiring the attacker to have a way of reading the PIN. This would certainly make an attack more difficult for **government**, but it would not stop it.

3. **ini/port**: have a USB or similar port on the indoors side of the lock, and initialize the lock by plugging the administrator's smartphone into it and continuing the setup from there. To circumvent this process, an attacker would have to already have compromised the administrator's smartphone, which defeats the point of trying to carry out **MITM onboarding**.

Table B.6 summarizes these options.

Table B.6: Security options for the SuperLock. Dollar signs indicate relatively how many resources needed to overcome a security option.

Option	Means				
	spoof user	**user commands**	**spoof admin**	**admin commands**	**MITM onboarding**
encryption[a]					
auth/password/vendor	\$[b]	\$[b]	\$[b]	\$[b]	\$[b]
auth/password/user	\$\$	\$\$	\$\$	\$\$	
auth/crypto/simple	\$\$\$\$	\$\$\$\$	\$\$\$\$	\$\$\$\$	
auth/crypto/local	\$\$\$	\$\$\$	\$\$\$	\$\$\$	
auth/crypto/global	\$\$\$[c]	\$\$\$[c]	\$\$\$[c]	\$\$\$[c]	
auth/biometric	\$\$\$	\$\$\$	\$\$\$	\$\$\$	
ini/first					\$\$\$
ini/PIN					\$\$\$[d]
ini/port					\$\$\$\$

[a]Does not prevent any means in itself. All **auth** and **ini** options rely on this.
[b]One exploit leaves all SuperLocks vulnerable.
[c]One exploit leaves all SuperLocks, and potentially many other devices, vulnerable.
[d]Adds a second factor over **ini/first**.

[15]The random PIN would be used as a pre-shared key for secure key exchange and subsequent encrypted communication.

Most of these options have many sub-options to address. The options presented here are high level, meant to capture the biggest tradeoffs. As described in the framework, sub-options that come up later will be examined by reapplying this framework.

These security options involve the following tradeoffs:

Security vs. features

None of the security options constrain SuperLock capabilities, although several do limit who can use it: **auth/crypto/local** requires users to have a CMS in their local OCF network, and **auth/biometric** requires users to have fingerprint scanners on their phones.

Security vs. security

Table B.6 contains most of the information needed to see the security tradeoffs between the security options. Beyond this, it should be noted that it would be possible to choose multiple **auth** options, using multiple factors to boost security. The **ini** options, on the other hand, are mutually exclusive.

Security vs. cost

auth/password/vendor is pretty cheap to implement and doesn't put any direct costs on any other parties.

auth/password/user does not take much more resources to implement than **auth/password/vendor**, and it doesn't put any direct costs on other stakeholders, either.

encryption does cost money to implement. Encryption is not easy, and merely choosing a reliable standard library to use takes time and expertise. Encryption also requires greater computing power, so the lock will be more expensive in itself, as carrying out encryption would be more computationally intense than anything else a SuperLock would have to do.

All **auth/crypto** options would cost substantially more to develop than the **auth/password** options. However, if there are standard tools and libraries to use, then these additional development costs drop substantially. They also require more computation than the **auth/password** options, but nothing beyond what **encryption** would already require.

auth/crypto/local rests upon a third party having built a CMS for people to use in their homes and people paying for them. However, as a general-purpose local CMS, SuperLock compatibility costs nothing more for either of these parties. Lockr developers only have to develop the interface to the local CMS. To do so, Lockr would have to tailor make a custom interface for any type of CMS they want SuperLocks to support, since the OCF specification does not specify how to connect to CMSs. This would cost slightly more than **auth/crypto/simple** if multiple CMS types are supported, because Lockr developers would need to make multiple interfaces that do the same thing.

auth/crypto/global would cost orders of magnitude more than the other **auth** options, because it requires, beyond mere development, the deployment and continuous maintenance of a global system. These costs could apply to either Lockr or a third party that supplies the global CMS. That said, if the third party provides a general global CMS, the marginal cost of supporting SuperLocks is essentially zero. As with **auth/crypto/local**, Lockr need only develop an interface to it. Unlike with **auth/crypto/local**, though, Lockr would only need to develop one CMS interface for **auth/crypto/global**.

auth/biomentric would also cost a lot to develop from scratch; even if the smartphone already supports fingerprint scanning, it would take a lot of upfront research and development to transform that information into a cryptographically secure and reliable authentication credential. However, a standard library would reduce these costs substantially.

ini/first would have negligible costs to any stakeholder. **ini/PIN** and **ini/port**, on the other hand, are by far the most expensive security options of all, as they require substantial hardware modifications that would raise the production costs of every SuperLock.

Security vs. usability

encryption involves a tradeoff between cost and usability. Investing more in computational power increases encryption speed and thus usability, but it makes the price of the lock go up. Users are probably willing to wait a moment for the lock to unlock, but users will become frustrated if the delay becomes non-negligible. Thus, the added security on **encryption** either requires a notable tradeoff with cost, with usability, or with a smaller combination of both of them.[16]

auth/password/vendor has a usability inversely proportional to the complexity of the password. It relieves the homeowners of the need to come up with a password, but that relative benefit could be smaller than the loss in usability if Lockr's password is more complex than the passwords that a user would set. Alternatively, **auth/password/user** comes at a high cost, especially when homeowners start to get a lot of IoT devices in their homes [30, 41]. The usability cost can quickly become irrational or incentivize users to start coming up with workarounds to their own security mechanisms [74].

By comparison, the **auth/crypto** options should be almost completely transparent to users, aside from whatever is involved with issuing and revoking them, because smartphones should be able to automatically choose the appropriate credential to authenticate with SuperLocks. Of these, **auth/crypto/local** would enable the local CMS to coordinate permissions and actions between SuperLocks and other devices, which, although it is not clear what devices one would want to coordinate with a door lock, could boost overall usability.

auth/biometric falls between the **auth/password** and **auth/crypto** options. Biometric-based authentication can be quick with modern smartphones, but it is slightly less reliable.

Since onboarding is only done once, the usability burden of all **ini** options is pretty tolerable. Although **ini/first** takes slightly less effort than the other two, few homeowners would feel inconvenienced by any of them.

B.1.5 Architectural Synthesis

Table B.7 shows the sections of the OCF specification and IoTivity documentation that provide guidance or requirements relevant to the security options and their trade-

[16]Determining where to settle between cost and usability for computation is outside the scope of PRSD, as security is constant anywhere along that spectrum.

offs. After accounting for the constraints of the OCF architecture, the Lockr engineers must choose **encryption**. In addition, they are left with the choice of one of the **auth/crypto** options and one of the **ini** options.

Notably, the OCF specification does not contain any useful information about choosing between credential types or about selecting, designing, or configuring a CMS or an interface to one.

Table B.7: Relevant documentation from the OCF specification [67] and IoTivity [46], and a description of their impact on the previously identified security options and their tradeoffs. All specification documents are from v1.1.0.

Documentation	Impact
Security 9.3.6	**auth/password** options not allowed
Security 9.3	**auth/biometric** implicitly not allowed
Security 9.3.3	Description of Asymmetric Authentication Key Credentials, suitable for **auth/crypto/simple**, **auth/crypto/local**, and **auth/crypto/global** and supports revocation
Security 9.3.5	Description of Certificate Credentials, suitable for **auth/crypto/local** and **auth/crypto/global** and supports revocation
Security 5	**encryption** and one kind of **auth/crypto** required
Security 11.2.3	Enumerates suitable ciphers for **encryption**, substantially reducing development costs
Security 5.1.1.2	Access Manager Service for **auth/crypto** boosts usability but decreases security (suitable for consideration as a sub-option)
Security 12.1	ACL guidance for **auth** is not specified, but will be in future versions of the specification
Security 7.3.4	**ini/first** specified and supported
Security 7.3.5	**ini/PIN** specified and supported
Security 7.3.8	**ini/port** supported
IoTivity	standard software implementation of **ini/first** and **ini/PIN**, substantially cutting down software development costs

B.1.6 Tradeoff Resolution

From the previous process, we already know that **encryption** must be selected as a security decision. Further, the OCF specification has details on how to implement this, and IoTivity does most of the heavy lifting already [47].

One of the **auth/crypto** options must be chosen. Compared to the others, **auth/crypto/global** costs a lot more for the developers or a third party to implement, and are susceptible as a single point of failure, and are in that way less secure.[17] Thus, it is worse than either **auth/crypto/simple** or **auth/crypto/local**.

auth/crypto/simple is more secure than **auth/crypto/local**. The difference may be small, but Lockr cannot be sure, because it cannot depend on local CMSs having any particular structure or configuration. Additionally, **auth/crypto/local** constrains the potential market and slightly raises costs over **auth/crypto/simple**. While using a local CMS does potentially offer some small usability benefits, this benefit does not offset the certain and uncertain downsides. Thus, we will choose **auth/crypto/simple** as a security decision.

Finally, between the **ini** options, **ini/first** must be chosen as a security decision. The other options are significantly more expensive. While **ini/first** is technically less secure, the only plausible attacker for any of the **ini** options is **government**, which none of them will stop.[18]

B.1.7 Security Sub-option Enumeration

This section demonstrates the process of reapplying PRSD to consider details of selected security decisions. To do so, we will consider two design issues that result from the previous decision to use **auth/crypto/simple**, which involves having SuperLocks themselves generate and manage asymmetric authentication key credentials. Specifically, we will investigate the enrollment process for guest devices, wherein guest devices receive the credentials needed to unlock the lock. Second, we examine the ACL management system.

To reapply PRSD towards these ends, developers need to consider how an attacker

[17]Even assuming perfect implementation, users would have to trust the global systems not to issue credentials to attackers, even though its known that the authorities in the existing CA for the web have done exactly that on numerous occasions [17].

[18]Lockr developers understand that, no matter what they do, a \$50–100 smart door lock is unlikely to thwart nation-state funded computer security activities.

might take advantage of these aspects of design to carry out any of the means from the threat model. From there, we proceed through Security Option Enumeration, coming up with options for mitigating these exploits.

Faulty enrollment could allow an attacker to simply receive legitimate credentials by, say, simply seeking enrollment, which would enable the attacker to impersonate an authorized user (**spoof user**.) It's even more important to protect requests for enrollment as administrators, as attackers with an administrator credential can impersonate an administrator (**spoof admin**.) In addition to preventing attackers from simply requesting credentials from the lock, it's important to prevent them from performing MITM on key transfer process, which would also allow them to carry out these same means.

There are multiple potential security sub-options for enrollment, with names prefixed with **enroll**, which more or less mirror the options for onboarding the SuperLock in the first place:

1. **enroll/first**: have the administrator command the SuperLock to enter an enrollment mode. The lock will trust the first device to seek enrollment in this time period. The devices could then exchange keys through ephemeral Diffie Hellman. Even if an attacker's device, and not the legitimate new user's device, is the first device to contact the SuperLock, the new user and administrator would recognize this and simply revoke the stolen credential and try again. This method would not stop credential theft through a MITM attack, though, although only **government** would ever be able and willing to do this.

2. **enroll/PIN**: use the same general procedure as **enroll/first**, but has the SuperLock send the administrator's phone a random PIN, which the administrator could tell the new user. This PIN would be used as a pre-shared key to create a secure channel using Diffie Hellman key exchange. This could mitigate MITM by requiring the ability to read the PIN, although **government** might still be able to do so.

Separately, poor ACL management would allow an attacker to modify ACLs in order to gain control of the lock without proper credentials, either at the user level (**user commands**) or administrative level (**admin commands**). From the previous Architectural Synthesis, we know that OCF supports two general approaches to managing ACLs:

1. **ACL/simple**: ACLs could be maintained on the lock itself, requiring attackers to exploit the lock itself to modify them.
2. **ACL/AMS**: ACLs could be maintained with an Access Manager Service (AMS), which coordinates ACLs for multiple devices on the local OCF network. The security of this approach depends on the security of the AMS.

These options are summarized in Table B.8.

Table B.8: Security sub-options for the SuperLock. Dollar signs indicate how many resources needed to overcome a security option, relative to the other sub-options on this list and the entries in Table B.6.

Option	Means				
	spoof user	**user commands**	**spoof admin**	**admin commands**	**MITM onboarding**
enroll/first	$$$		$$$		
enroll/PIN	$$$[a]		$$$[a]		
ACL/simple		$$$$		$$$$	
ACL/AMS		$$$[b]		$$$[b]	

[a]Contains a second factor over **enroll/first**

[b]Depends on reliability of the AMS. Potential central point of failure to all devices on local network.

These security sub-options have the following tradeoffs:

Security vs. features

No sub-option constrains a SuperLock's capabilities. However, **ACL/AMS** does require SuperLock owners to own and manage an AMS in their own homes.

Security vs. security

Table B.8 illustrates most of the security tradeoffs between these sub-options. It is also worth noting only one sub-option can be chosen from each category.

Security vs. cost

enroll/first would not have significant marginal development costs, as the necessary cryptographic routines are already supported by the standard cryptography libraries that SuperLocks were already planned to use anyway. The rest of this option should be relatively cheap to develop. This option puts no costs on other stakeholders.

enroll/PIN requires all the same development as **enroll/first**, and it requires Lockr to develop the scheme for sharing the PIN and erecting a secure channel from it; however, standard cryptography libraries will do most of the heavy lifting here.

If the lock uses **ACL/AMS**, then the lock developers will not have to roll their own ACL management system for the lock, so the AMS sub-option is cheaper to develop than the local ACL management sub-option. Furthermore, the marginal costs that **ACL/AMS** imposes on other stakeholders is negligible, assuming the AMS is already there to begin with.

Developing **ACL/simple** would certainly cost Lockr money, since the OCF specification provides no guidance for doing it (see Table B.7).

Security vs. usability

enroll/PIN is slower and more cumbersome than **enroll/first**, as two users would have to manually share a key suitable for establishing a secure channel. However, it is worth noting that the overhead for both of them is only one-time per user, probably pretty intuitive, and only takes a few seconds.

A well designed AMS would be equally or slightly more usable than using **ACL/simple**. For the most part, ACL management in general should be transparent to admins and users. However, an AMS could automatically set access policies for other home devices for new guest users of the lock. For instance, hypothetically, an AMS could automatically allow new Airbnb guests to control a smart TV and the lights but not the thermostat (and then revoke those permissions when the guest checks out.) However, such automation is beyond the control of the company producing the lock and thus is not guaranteed.

B.1.8 Architectural Synthesis on Sub-options

Notably, the OCF specification does not include any information about bringing guest devices on to a local OCF network in general, let alone discusses the security considerations of such actions. While there is general discussion of “onboarding” and initial configuration (for example, see Section 11.2 of the Core Specification v1.1.0) and endpoint discovery (Section 10 of the Core Specification v1.1.0), the specification is silent about prescribing a particular way to introduce new devices; it only specifies certain state attributes that must be achieved by the end of it [67]. Likewise, the Security Specification only discusses options for setting up new devices to be owned by the local network, which would not work for guest devices [67].

With that in mind, Table B.9 summarizes all the aspects of the specification that do relate to the sub-options under consideration.

Table B.9: Relevant documentation from the OCF specification [67] and IoTivity [46], and a description of their impact on the previously identified security options and their tradeoffs. All specification documents are from v1.1.0.

Documentation	Impact
Core 11.2	General discussion of “onboarding” and initial configuration, which is the closest the specification gets to guest enrollment, let alone the **enroll** options
Security 5.1.1.2	**ACL/AMS** boosts usability but decreases security over **ACL/simple**
Security 13.5.1	specification of ACL resources, making **ACL/simple** possible

B.1.9 Sub-option Tradeoff Resolution

For enrollment, using the random PIN to verify identity bolsters security over **enroll/first** by making it more difficult for attackers who have already compromised the local network to MITM the enrollment process. While this is certainly a security advantage, it would not stop **government**, which is the attacker that would have the resources and motivation to do this, anyway. Furthermore, these attacks are, in general, extraordinarily unlikely. This marginal security gain of **enroll/PIN** probably is not worth the usability and cost tradeoffs it incurs, however. Thus, **enroll/first** will be chosen as a security decision.

Storing ACLs locally with **ACL/simple** primarily protects against hacktivists and botnet-dealing organized crime seeking one more bot to control. Its tradeoffs compared to **ACL/AMS** are mixed. Requiring an AMS limits the market to people who already have one; however, doing so saves substantial development costs—thanks to the OCF specification's lacking ACL management guidance. Additionally, **ACL/AMS** could increase usability by automatically setting up new Airbnb guests with ACLs for other devices. If the OCF specification was more clear, this decision would be much simple, but the uncertainty of the security of AMSs, combined with the uncertainty of managing ACLs, makes this decision difficult. Since Lockr is a profit-driven company, the difference would probably come down to the particulars of how much of their potential market already have AMSs versus how much more it would cost to develop ACL management themselves.

Thus, the final security decisions for this case study are **encryption**, **auth/crypto/simple** with **enroll/first** and either **ACL** option, and **ini/first**.

B.1.10 Discussion

This case study demonstrates that PRSD can be useful for thinking critically about the tradeoffs involved when weighing options in a home IoT product's security design. At one point or another, all elements of each process were used towards ultimately deciding between competing security options. Thus, if PRSD called for any additional analysis over what the developers at the hypothetical company Lockr would have done otherwise, it would only be because the developers were glossing over important factors in the analysis.

More specifically keeping the ties between security options, means, attackers, and harms in mind made it more straightforward to weigh what security value was gained and lost between options than simply considering the security options in isolation. Consequently, it was easier to compare the relative security tradeoffs with other factors like capabilities, cost, and usability. By comparison, traditional security frameworks do consider security options in isolation and rarely have developers compare security impacts explicitly against other tradeoffs.

It's also clear that architectural constraints can do a lot to resolve tradeoffs (in full or in part) before the actual tradeoff resolution process. Architectural constraints eliminated all of the password-based security options and required the encryption security option, going so far as to specify particular cipher suites to use for this application. Furthermore, it narrowed the list of eligible authentication security options.

This framework also laid bare several gaps in the current OCF specification. Notably, there is a dearth of specification or guidance for choosing an implementing a Credential Management System, ACLs, and securely bringing guests on to the network. If this analysis had called for using a full CMS (as opposed to the ad hoc local credential generation option that was chosen), the sub-options would have been enormously complicated: misplaced incentives and poor oversight can lead even technically sound CMSs to issue legitimate certificates to attackers. It might be outside the scope of the specification to define CMSs in full, but it would still be reasonable for the specification to define interfaces for safely interacting with CMSs or even provide some security guidance about choosing or administering them.

Likewise, the ACL management section of the specification is explicitly empty, aside from a note that it will be filled in sometime in the future. The details of ACL management are more fine grained than this initial use of the framework called for; however, considering those sub-options could prove extremely difficult. It is notoriously difficult to implement and manage ACLs correctly [89].

The lack of specification for guest devices was neither acklowedged nor excusable. The Core Specification implicitly assumes that there are always carefully crafted ACLs for determining what resources can be shared with a guest.[19] However, the Security Specification does not include any information for recognizing or generating ACLs for guest devices and their credentials; its section on security provisioning (section 7 of v1.1.0) assumes that new devices are unowned and will become owned by the local network. Guest devices, on the other hand, will already be owned by a

[19]Actually, it says that there should be as few unsecured resources as possible.

separate network, and it's unlikely that house guests will want to completely reset their portable devices twice every time they want to interact with their hosts' devices (first time to join the host's network, second time to re-join the guest's own network.) Thus, vendors are left to roll their own nonstandard and probably risky ways of having their devices interact with guests (or act as guests.)

B.2 Updating for Airbnb Integration

B.2.1 Given

Picking up where the previous example left off, let's suppose that Lockr, having completed the initial PRSD analysis, decides that it wants to integrate Airbnb with SuperLocks. That is, they want the lock to check for Airbnb guests, send them the appropriate keys before they arrive, and revoke them after they leave. Ideally, this will all be automated yet also allow for homeowners to override the automatic functionality however they want.

We will assume that the company decides to design for these features immediately after completing the analysis from the previous example and that there is no other new information that must be considered.

Lockr knows about Airbnb Host Assist, a service designed by Airbnb to facilitate "easy key exchange" and "keyless entry" that is largely automated [4]. This is the service they intend to use. They realize that they are competing with existing smart locks that also use Host Assist, such as August Locks [14]. The startup is hoping that OCF compliance will provide will provide a competitive edge, as it will allow for homeowners and guests to easily integrate their locks into the rest of their home automation and alleviate the need to install and learn custom software designed just for the locks.[20]

The startup does not want to remove the previously designed manual method for

[20]The OCF specifications, being a communications standard, ideally allows all OCF devices to communicate with each other in a more or less uniform manner. Thus, users should be able to interact with most OCF devices the same way, rather than having to download a separate app for each one.

sharing keys, as they realize that some users will have guests that are not Airbnb guests.

B.2.2 Situational Modeling

Capabilities

This update does not change the outcome capabilities of the lock. It does, however, require the additional support capabilities necessary to have full Internet connectivity in order to communicate with Airbnb through the Host Assist API, which implies communication with non-OCF compliant Airbnb servers.

Table B.10: Capabilities of the SuperLock after integrating with Airbnb. New capabilities over those in Table B.1 have *emphasis*.

Capability	Description
lock	lock and unlock mechanism
communication	Communication with phones with the OCF standard.
update	receive Internet updates over a proprietary protocol
physical lock	backup physical lock and unlock mechanism
reset	reset button restores SuperLock to factory settings
Internet	*Communicates with arbitrary Internet traffic to facilitate communications with Airbnb*

Environmental models

These new capabilities require the lock to be able to deal with whatever traffic it could receive from the Internet. They are also dependent on the Host Assist service, which may have dynamic requirements.

With Airbnb integration, the locks will depend on third party services. This can make responsibility and liability complicated. In general, if developers are aware of relevant policies in jurisdictions that the locks might fall under, it would be prudent to include them in the environmental models. However, since the liability in this situation is dynamic and uncertain, there is nothing specific to include.

Stakeholder models

Table B.11 summarizes the original and new stakeholders, which accounts for changes in stakeholders and dependent assets that result from Host Assist integration.

It is worth noting that, since Airbnb itself is not expected to put any effort towards SuperLocks specifically, their entries for "Expertise" and "Resources" are accordingly empty.

Table B.11: Stakeholder Models. Changes over Table B.2 have *emphasis*.

Type	Stakeholders	Interests and Assets	Expertise	Resources
Users	homeowners, Airbnb guests	safety, privacy, valuables, OCF network, availability, *Airbnb accounts*	minimal	minimal
Admins	homeowners	safety, privacy, valuables, OCF network, availability, *Airbnb accounts*	minimal	low
Developers	Lockr engineers	company revenue	high[a]	full time job
Support	Lockr engineers	company revenue	high[a]	full time job
Third parties	*Airbnb*	*company revenue*		
Public	People who use the Internet	Internet access		

[a]Above average security expertise, compared to other developers.

B.2.3 Threat Modeling

Integrating with Airbnb will not remove any threats, as no capabilities will be removed from SuperLocks. To perform threat modeling, then, the Lockr developers would only need to examine how this new functionality would expand the existing threat model.

Harms

The new Airbnb integration could allow an attacker to cause additional harms through the users' Airbnb accounts. Depending on the specifics of Host Assist (which are not public and subject to change), this could allow an attacker to glean sensitive personal information or to change the configuration of one's Airbnb account. Notably, an attacker might be able to change a user's payout account or possibly steal credit card information.

This expansion of harms is summarized in Table B.12.

Table B.12: Potential harms resulting from SuperLocks. Additions over information in Table B.3 has *emphasis*.

Harm	Description	Assets at risk
break in	attackers get into the house	safety, privacy, valuables, company revenue
pivot harm	the SuperLock is exploited and used as a proxy to devices on the OCF network	OCF network, company revenue
damaged	the lock is damaged	safety, privacy, valuables, company revenue
bot	the SuperLock is incorporated into a botnet	Internet access, time, company revenue
account compromise	*an attacker could compromise a user's Airbnb account*	*Airbnb accounts, company revenue*

Attackers

Table B.13 shows how Airbnb integration would affect the potential attackers. The only attacker whose interests significantly change with Airbnb integration is **expansive crime**, who could stand to gain from stealing credit card information at scale. One can imagine that **pranksters** and **hacktivists** may also enjoy Airbnb account access, but these scenarios seem far fetched and unlikely, so we do not assign them high expected interest.

Means

Although Airbnb integration through Host Assist falls outside of the OCF specification, as Airbnb's servers are not OCF-compliant. As explained in section 5.5 of the OCF Core Specification v1.1.0 [67], OCF clients like the SuperLock communicate with non-OCF devices by creating a mapping between non-OCF data or constructs and OCF resources. Section 5.5 outlines one way to implement this. Since SuperLocks will have to interpret non-OCF Internet traffic as OCF resources, then, non-OCF traffic could take advantage of this interpretation to potentially cause harms, which would be a valid means within the scope of PRSD.[21]

[21]The proprietary update scheme is not treated this way, because updates are unlike general

Table B.13: Potential attackers with an interest in SuperLock harms. Exclamation points after harms indicate relatively higher expected interest in causing that harm. Changes over those in Table B.4 have *emphasis*.

Attacker	Description	Harms of interest
burglars	criminals wanting to commit burglary	**break in** (!!), **damaged**
prankster	people looking to cause mischief	**break in**, **locked out**, **damaged**, **pivot harm**, ***account compromise***
government	agents of law enforcement or espionage	**break in**[a], **damaged**, **pivot harm**[a]
quarreling	users fighting with each other	**locked out**
seizure	landlords or other authorities seizing property	**locked out**
hacktivists	activists who make statements through hacking	**bot** (!!), ***account compromise***
expansive crime	organized crime looking to steal data or amass botnets	**pivot harm** (!), **bot** (!!), ***account compromise*** *(!!)*

[a]States will have an strong interest in causing this harm to certain suspects or activists, but these cases are extremely rare, and those users will likely know the risks.

Table B.14 reflects the new categories of means enabled by the new capabilities. In either case, the exploit depends on taking advantage of however the SuperLock will map non-OCF traffic to OCF resources and operations; the difference is that one takes advantage of particulars of the Host Assist API (which are uncertain and subject to change), and the other takes advantage of how the mapping algorithm interprets Internet traffic in general.[22]

Table B.14: Means through which attackers could cause harms by leveraging SuperLock OCF communications. Changes from Table B.5 have *emphasis*.

Means	Description	Harms
spoof admin	impersonating an administrator	**break in**, **damaged**, **pivot harm**, **bot**, ***account compromise***
admin commands	sending commands that entail administrator-level actions	**break in**, **damaged**, **pivot harm**, **bot**, ***account compromise***
spoof user	impersonating an authorized user	**break in**
user commands	sending commands that entail user-level actions	**break in**
MITM onboarding	man-in-the-middling the initial onboarding process	**break in**, **damaged**, **pivot harm**, **bot**, ***account compromise***
malicious Airbnb traffic	*maliciously exploiting the Host Assist API to exercise unauthorized control of the loca*k	***break in, damaged, pivot harm, bot, account compromise***
malicious Internet traffic	*maliciously sending general Internet packets to the lock to exercise unauthorized control of the lock*	***break in, damaged, pivot harm, bot, account compromise***

communications in that the OCF specification does not discuss them at all, so exploits of the update system are still out of scope.

[22]These means highlight a notable interaction between implementation and design. While recognizing that implementation can be both important and difficult, PRSD limits its scope to design and thus has to ask developers to temporarily assume that their implementations will be correct. However, the same cannot be assumed of Host Assist and general Internet traffic, whose implementations are beyond Lockr's control. Thus, these new means reflect Lockr's limited trust of originators of Host Assist and general traffic.

B.2.4 Security Option Enumeration

The previous security decisions, which include **encryption**, **auth/crypto/simple** with **enroll/first** and an **ACL** option, and **ini/first** (see Section B.1.9), do not need to change. The only new harm enabled by the old means is **account compromise**, but no attacker is so motivated to cause this harm that it changes the tradeoffs between security options enough to change a previous security decision. Thus, the previous analysis can be left as is.

With that, only the new means—**malicious Airbnb traffic** and **malicious Internet traffic**—need to be addressed in this process. These means are not addressed by the existing security decisions, so it is necessary to identify new security options.

Security options

It's important to note that Lockr does not have control over the security design of every component of the system. Airbnb controls their own servers and Host Assist itself. Lockr controls its own translation of Host Assist traffic to OCF resources, how it interacts with those resources, and verification and validation [48] of traffic it receives.

Although the lack of details about Host Assist make it difficult to reason about what security measures are needed, we will assume that Airbnb does at least attempt to implement some basic authentication and integrity measures.

With that in mind, the startup has the following security options:

1. **whitelist**: design the lock so that the only non-OCF traffic it accepts is well validated, sanitized, and protected by any authentication and integrity measures Host Assist offers. All other non-OCF traffic is dropped. Assuming proper validation and verification of the algorithm that translates between the Host Assist API and OCF resources, as well as proper implementation, this completely protects against **malicious Internet traffic**, and **malicious Airbnb traffic** will be as secure as Host Assist allows for.

2. **traffic review**: for each connection request, send traffic summaries to homeowners to manually evaluate and allow. This could allow homeowners to monitor

and protect their locks with confidence, and well-designed notifications could allow a savvy homeowner to protect against almost any threat. However, most homeowners will not be savvy enough to make these decisions wisely, and this option provides no security for apathetic or ignorant users.

3. **no Host Assist**: do not do Host Assist, which completely protects against **malicious Internet traffic** and **malicious Airbnb traffic**.

Implicitly, choosing none of these options is also an option.

These options are summarized in Table B.15.

Table B.15: New security options for the SuperLock. Dollar signs indicate relatively how many resources needed to overcome a security option.

Option	Means	
	malicious Airbnb traffic	**malicious Internet traffic**
whitelist	\$\$\$[a]	\$\$\$\$
traffic review	\$[a]	\$
no Host Assist	\$\$\$\$	\$\$\$\$

[a]Only as secure as the Host Assist API can be

Tradeoffs

These tradeoffs may depend somewhat on legal responsibility and liability between Lockr and Airbnb. However, as noted in the environmental models described in Section B.2.2, the policy situation here is dynamic and unclear. Without having a stable policy environment, the developers will need to consider how all possibilities would affect the tradeoffs.

Security vs. features

Using **no Host Assist** entails entirely abandoning Airbnb integration. It's worth noting that this option could make the lock less competitive against other smart locks that do support Host Assist; the **no Host Assist** SuperLock might be more secure, but it's hard to prove this to consumers. On the other hand, implementing Host

Assist opens another avenue of attack but enables all the features described in the capabilities section.

Security vs. security

Table B.15 contains most of the information needed to see the security tradeoffs between the options. Beyond this, it is worth noting that **no Host Assist** is mutually exclusive of **whitelist** and **traffic review**.

Legal responsibility and liability, which are uncertain at this time, would greatly affect Lockr's interest in providing these security options.

Security vs. cost

no Host Assist costs nothing, and saves Lockr from the development costs of implementing Host Assist integration. However, having Airbnb integration could potentially increase sales enough to cover the extra development costs.

The **whitelist** probably would not cost much over the existing cost of implementing Host Assist integration at all. The only additional requirements are to have the algorithm that translates between Host Assist and OCF resources to perform careful input validation. This validation probably is not too difficult, presuming that Airbnb defines the Host Assist API well.

The **traffic review** would cost the most of the three options to develop, as it is a significant and nontrivial standalone feature. Details of this cost would depend significantly on the details, which will be considered if this option is chosen as a security decision.

Security vs. usability

The **whitelist** imposes no usability burden on either homeowners or guests over not having it. The **traffic review**, on the other hand, would require extra manual effort every time the lock attempts to communicate with Airbnb. This extra manual effort largely nullifies any of the convenience and automation that makes Airbnb integration appealling in the first place. Similarly, opting for **no Host Assist** means sacrificing all of the usability benefits of Airbnb integration.

B.2.5 Architectural Synthesis

The OCF specification does not provide any security guidance for Airbnb integration, which is understandable because Host Assist does not comply with the OCF specification. The Open Connectivity Foundation intends for OCF specification to abstract a wide variety of underlying protocols, but Host Assist is not one of them. Thus, it is up to Lockr and Airbnb to make their own security decisions here.

B.2.6 Tradeoff Resolution

The **traffic review** option provides dubious security at high cost and largely counteracts many of the usability advantages of having Airbnb integration in the first place, so it is not worth implementing.

Compared to having Airbnb integration without it, the **whitelist** option seems like a clear choice. It almost entirely protects against **malicious Internet traffic** and goes a long way towards protecting against **malicious Airbnb traffic**, has no usability costs, and imposes only a small additional cost on the developers.

The nontrivial decision is between **no Host Assist** and **whitelist**. Since Host Assist falls outside of the OCF specification, translating between that API and OCF resources could cost a lot to develop and is subject to error. Furthermore, the **whitelist** can only protect against **malicious Airbnb traffic** as much as Host Assist allows—if the authentication features of Host Assist are unreliable, for instance, then it is difficult for the lock to validate against spoofed traffic no matter what the developers do.[23] On the other hand, having a **whitelist** instead of **no Host Assist** allows for all the features and usability afforded by Airbnb integration.

For the developer's own calculus, the decision rests largely on the relevant regulatory environment. If there is stable regulation that clearly puts liability for problems with Host Assist itself on Airbnb, then the startup would only be responsible for

[23]It's worth noting that, if Host Assist complied with the OCF specification, then developers would be free to use any of the OCF security features, such as ACLs and credential management, to ensure the lock's security regardless of Airbnb's behavior. Additionally, being OCF compliant would cut down on development costs significantly.

harms caused by **malicious Internet traffic**, which **whitelist** protects against almost entirely. In this case, from the startup's profit-driven perspective, **whitelist** is the clear choice if it can afford the development costs.[24] Otherwise, the startup might be on the hook for Airbnb's lackadaisical security design decisions, which is a risk that probably is not worth taking. In this situation, it is better to abandon Airbnb integration and choose **no Host Assist**.

B.2.7 Discussion

This case study demonstrates how PRSD makes it relatively straightforward to update existing analyses to account for new information. Much of this case study is copied from or directly derives from the previous analysis, and it was simple to determine what aspects of the previous analysis needed reconsideration at all. The methodical design of the framework also made it easy to determine where to incorporate new information.

This case study also demonstrates how PRSD operates at the boundaries of its scope and at the boundaries of the control of the agent employing it. The framework still facilitates systematic and critical decision making when considering system components that act across the boundary of the scope (here, communication between an OCF device and a non-OCF server.) Also, this case study shows that the framework exposes boundaries of security control between different components of an IoT system (here, the boundaries between Host Assist and the lock's OCF representation of it.) In general, the framework is able to do this because it has the agent consider the threat model as thoroughly as possible before having the agent consider what security options she has to mitigate them.[25]

Additionally, this case study shows, through considering the interplay between Host Assist and the OCF specification, the great effect that architectural standards

[24]This is not to be confused with the socially optimal choice. If Host Assist itself is sufficiently insecure and Airbnb does not compensate for harms caused by **malicious Airbnb traffic**, then this might not be the socially optimal choice.

[25]As an aside, highlighting these boundaries of control might help to reason about what parties *should* have what responsibilities in an IoT system.

can have on security decision making. The proprietary Host Assist gives Airbnb a huge amount of control over the security options that are possible in this scenario; if Host Assist was OCF compliant, then it would have to allow the lock to use its own ACLs to make its own trust decisions. Moreover, this demonstration serves as an example of how the framework can be used as a tool to reason about the different interfaces between architectures and their security effects in particular scenarios.

Finally, this case study shows that the regulatory environment, or even instability of it, can have a decisive effect on the developers' calculus when weighing security options. Although unsurprising, it is still worth noting.

B.3 Voice-controlled Hub

B.3.1 Given

Let's suppose there's a moderately-sized company (not a new startup, but not a tech giant) called *OCFMakers* wanting to design, create, and market an OCF compliant device that acts as a simple-to-use hub, henceforth called the Hub, for interacting with and coordinating actions between other OCF-compliant devices throughout a home. It also acts as something like a personal assistant for homeowners.

The Hub primarily takes voice commands as input and provides audio output. It sends the raw sound signal to cloud services owned by OCFMakers, and these servers extract machine-readable commands from the raw sound signal of the voice command, including a voice signature (for identifying the speaker) and any other metadata that OCFMakers wants to note. The cloud services store this information and send a "command tuple" including a machine-readable command, voice signature, and metadata back to the Hub that recorded it. This transmission all occurs over the OCF specifications.

From there, the Hub matches the command tuple to a table which determines where to send it, and the command—not the whole command tuple—is sent to the appropriate OCF servers, which include cloud servers, local OCF devices, smartphones, etc. Upon sending a command, the Hub will wait and listen for an acknowledgement as well as an indication of whether or not to listen for follow up instructions. All these communications follow the OCF specification.

For instance, a person might tell a Hub, "Was there any suspicious activity last night?" The Hub would send this raw input to OCFMakers' cloud services to process, and the service would return the tuple of command, voice signature, and metadata. Using some sophisticated proprietary matching algorithm, the Hub would determine that this is the sort of command to send to the local OCF security camera. Accordingly, the Hub sends the command to the camera, which replies that it received the command and is crafting a response to relay. The camera does its own processing and decides to tell the Hub to respond with "All is well. Nothing to report from last night." The Hub receives this final response and announces this response to the user.

Hubs store minimal local information and handle few requests through their own processing. While it may be able to self-report details of its configuration through locally processed commands, even common small tasks like managing calendars and todo lists are offloaded to cloud servers or other OCF devices. However, it does store whatever identification tokens, credentials, and other account information is necessary for it to fulfill its basic functions.

OCFMakers plans to have its own servers to support, in addition to voice signal processing, basic personal assistant operations, and they are in talks with various other platforms to support more functionality from the get-go, such as music streaming and weather services. OCFMakers also plans to publicly release details for integrating with its voice-command pattern matching algorithm. With this, the Hub will be able to interoperate with any other compatible OCF products that provide instructions for handling relevant voice commands.

Ideally, the Hub's only button will be a power switch. Also, it will need some sort of web app administrative interface capable of handling setup and configuration that is too advanced for the voice interface.

The Hub will be marketed as a competitor to the Amazon Echo [5]. OCFMakers hope that OCF compliance and interoperability with other OCF compliant devices will give them a competitive edge.

OCFMakers also plans to provide seamless software updates over the Internet with a completely proprietary scheme.[26]

To be more specific, Hubs also record state information, such as context in a conversation, and use the command tuple and state to determine which information to send to what services or devices.[27] When discovering services and devices, OCF devices will need to tell the Hub what commands to send their way. State is stored as the last several commands issued in the active session. It is up to the devices and services that actually process the commands to make sense of what the state is from those commands.

The Hub will support guest users insofar as connected services also support commands without a known voice signature.

B.3.2 Situational Modeling

Capabilities

From the given information, we can discern several discrete Hub capabilities. These are listed in Table B.16.

Table B.16: Hub capabilities

Capability	Description
voice input	sensing and processing of voice input, including the extraction of voice signatures
audio	audio output
communication	send OCF communications over the local network and the Internet
web app	serve a web application for administration
credentials	store and manage user credentials
command match	algorithm for routing command tuples to the services that need them
update	receive Internet updates over a proprietary protocol

[26]The update scheme itself is outside the OCF specification, as it does not cover updates in general.

[27]It's unclear at this time how much metadata will be necessary for this capability, but OCFMakers would like to have the option of collecting whatever metadata they want.

Environment

Hubs will likely remain more or less stationary within a home. They will probably be used almost exclusively indoors within people's homes.

They will likely connect to almost all other OCF devices in the home they are set up for. They will also need to communicate with OCF servers outside of the home they are set up, including phones and cloud servers. Without timely access to OCFMakers' voice processing cloud services, a Hub cannot operate.

They will also need to interact with known users and unknown guests.

Stakeholders

Table B.17 details the relevant stakeholders.

Table B.17: Stakeholder Models

Type	Stakeholders	Interests and Assets	Expertise	Resources
Users	homeowners, guests	Hub, OCF network, data, credentials, availability, trust	minimal	minimal
Admin	homeowners	Hub, OCF network, data, credentials, availability, trust	minimal	low
Developers	OCFMakers engineers	company revenue	medium	full time job
Support	OCFMakers engineers	company revenue	medium	full time job
Third parties	Hub-compatible product developers	company revenue, IoT products, data		
Public	People who use the Internet	Internet access, time		

Several aspects of these stakeholders require explanation. Here, the "OCF network" asset includes the local network itself, the connected devices, and any remote OCF servers a Hub may contact. "Availability" refers to users' and admins' abilities to use Hubs. Similarly, "trust" refers to users' and admins' trust in their respective Hubs, including the answers to their queries.

Admins are a subset of users: they are the homeowners who spend a bit of extra effort setting up and managing their Hub. This extra effort is reflected by the difference in the “Resources” column. Similarly, Developers and Support are both engineers at OCFMakers. However, since OCFMakers is an established company, it probably has a separate division that maintains the cloud services that keep Hubs running.

Third parties are the vaguest stakeholder group, as they include any group that makes Hub-compatible devices. In addition to their company revenue, the IoT products they product and the data those products collect could also be affected by Hubs.

The public could be affected by Hubs if they are incorporated into a botnet. As part of a botnet, Hubs could either carry out DDoS attacks, which would hinder Internet access, or send spam, which would waste the public’s time. We will examine this more in the Threat Modeling section.

B.3.3 Threat Modeling

Harms

There are multiple harms that could result directly from Hubs, indirectly from Hubs, and to Hubs themselves. These harms are detailed in Table B.18.

Attackers

Table B.19 explains the categories of attackers interested in the harms enabled by the Hub. Attackers are listed roughly in order of increasing security expertise and resources.

Small crime differs from **expansive crime** primarily by a qualitative difference in size and organization. **Small crime** could be an individual or a small group of people who cause security breaches more or less informally. Brian Kreb’s description of Anna-Senpai [52], for instance, would fit this category. **expansive crime**, on the other hand, would include larger organizations with more professional organization and expertise; stereotypical Eastern European criminal hacking organizations would fit this category. While both attackers can cause devastating harm, **expansive crime** will tend to have more patience, security expertise, and funding to develop more sophisticated attacks. In both cases, these attackers are almost always interested in

Table B.18: Potential harms resulting from Hubs

Harm	Description	Assets at risk
unavailable	the Hub ignores commands	OCF network, availability, company revenue
false response	Hub responds to command falsely	data, availability, trust, company revenue
replace	Hub is damaged and must be replaced	Hub, availability, company revenue
bot	Hub incorporated into botnet	Internet access, time, company revenue
credential theft	credentials for other accounts, devices, or services stolen	credentials, company revenue
pivot harm	the Hub is exploited and used as a proxy to cause harm through other devices on the OCF network	OCF network, IoT products, data, company revenue
pivot bots	the Hub is exploited and used as a proxy to incorporate other devices on the OCF network into botnets	Internet access, time, company revenue
i/O theft	voice commands and their responses stolen	data, company revenue

Table B.19: Potential attackers with an interest in Hub harms. Exclamation points after harms indicate relatively higher expected interest in causing that harm.

Attacker	Description	Harms of interest
prankster	people looking to cause mischief	**unavailable**, **false response**, **replace**, **credential theft**, **pivot harm**
hacktivists	activists who make statements through hacking	**unavailable** (!), **false response** (!), **replace** (!), **bot** (!), **credential theft** (!), **pivot harm** (!), **pivot bots** (!), **i/O theft** (!)
small crime	small-time criminals looking to steal data or amass botnets on a budget	**bot** (!), **credential theft** (!), **pivot harm** (!), **pivot bots** (!)
expansive crime	organized crime looking to steal data or amass botnets	**bot** (!!), **credential theft** (!!), **pivot bots** (!!)
APT	highly motivated, well-funded, expert, highly targeted, and patient attackers	**credential theft**[a], **pivot harm**[a], **i/O theft**[a]
government	agents of law enforcement or espionage	**credential theft**[b], **pivot harm**[b], **i/O theft**[b]

[a]APT will have an strong interest in causing this harm to a very small number of special individuals, and those targeted users will likely know the risks.

[b]States will have an strong interest in causing this harm to certain suspects or activists, but these cases are extremely rare, and those users will likely know the risks.

stealing valuable data and credentials or building botnets, so their interest in any single target is low.[28]

Means

There are a variety of means within the OCF scope that attackers could employ to cause the harms made possible by the Hub. Table B.20 specifies them.

Table B.20: Means through which attackers could cause harms by leveraging SuperLock OCF communications.

Means	Description	Harms
spoof admin	impersonating an administrator	**unavailable**, **false response**, **replace**, **bot**, **credential theft**, **pivot harm**, **pivot bots**, **i/O theft**
admin commands	sending commands that entail administrator-level actions	**unavailable**, **false response**, **replace**, **bot**, **credential theft**, **pivot harm**, **pivot bots**, **i/O theft**
MITM onboarding	man-in-the-middling the initial onboarding process	**unavailable**, **false response**, **replace**, **bot**, **credential theft**, **pivot harm**, **pivot bots**, **i/O theft**
eavesdrop	listening in on communications between the Hub and another device	**credential theft**, **i/O theft**
modify traffic	actively modify OCF communications between a Hub and another endpoint	**unavailable**, **false response**, **pivot harm**
spoof device	pretend to be a legitimate endpoint that connects to the Hub	**unavailable**, **false response**, **credential theft**, **pivot harm**, **i/O theft**
overload commands	assert that a malicious device is the proper recipient of a voice command intended for another device	**unavailable**, **false response**, **credential theft**, **i/O theft**

As with the SuperLock, **spoof admin** differs from **admin commands** in that the

[28]The exception would be that **small crime** might occasionally want to burglarize a home with the help of **pivot harm**.

former exploits some weakness in identification and authentication and the latter exploits some weakness in access control. In either case, we assume that administrator-level access could enable an attacker to get root on a Hub.[29]

Overload commands lies at the edge of the scope of PRSD. Determining which commands to route to which devices is the job of the proprietary pattern matching algorithm, which itself is not affected by the OCF architecture. However, the structure of the OCF architecture may enable an attacker to convince a Hub to route commands intended for another device to a malicious device. In this sense, this means is in scope as a separate issue than simply **eavesdrop** or **spoof device**.

There are some notable means that are not within the scope of PRSD. One user could impersonate another user's voice, causing harm by acting as the impersonated user for the Hub. However, this attack is completely outside the consideration of the OCF architecture, so it does not merit consideration here. For the same reason, any attack that would interfere with the proprietary scheme that Hubs use to receive software updates cannot be considered here.

B.3.4 Security Option Enumeration

Security options

Here, we present a list of high-level security options that a company like OCF-Makers may actually consider. As with all other security designs, it is infeasible for OCFMakers to think of, let alone fully consider, every potential option for mitigating threats to Hubs. These options do, however, cover every means.

encryption: use a strong encryption scheme to make all OCF communications confidential. All other security options rely on this option, and (assuming a smart implementation with proper key exchange, etc.) it would prevent eavesdropping on OCF communications almost entirely.

There are several authentication schemes that could allow a Hub to verify the identities of administrators and potentially OCF endpoints. For the time being, we assume some sort of permission scheme (such as access control lists) to control access

[29]See Section B.1.3 for a justification of this assumption.

to the Hub's resources. Authentication itself addresses **spoof admin**, **spoof device**, and **modify traffic**, and the permissions system that accompanies it will mitigate **admin commands**. These options, whose names start with **auth**, are:

1. **auth/password/vendor**: use a single manufacturer-set password listed in each Hub's manual to control administrative access to the web application. This would slightly impede **spoof admin** and **admin commands**, although any slightly motivated attacker could figure out this password by grabbing a copy of the Hub instruction manual.

2. **auth/password/admin**: have the administrator set her own password during the initial setup process. This could slow attackers' ability to **spoof admin** or send **admin commands**, but would probably would only seriously impede **pranksters**, some **hacktivists**, and some **small crime**.[30]

3. **auth/crypto/local**: use cryptographically secure public key credentials to authenticate administrators, other devices, and services, and store the credentials locally. By default, the Hub itself would store these credentials, although OCF-Makers could later explore options for delegating credential storage to another local endpoint. Devices would need to present their credentials to the Hub for authentication. This method would be extremely difficult to defeat, as it would require either stealing the credential, compromising an OCF endpoint, or breaking public key cryptography—methods typically only available to **APT** and **government**.[31]

4. **auth/crypto/remote**: use a single CMS to generate, issue, and revoke the credentials for all Hubs and compatible devices. This provides the same protections as **auth/crypto/local**, except that the CMS could become a central

[30]Passwords can be guessed or socially engineered. Most people do not choose hard-to-guess passwords, even with most complexity rules [3]. Indeed, researchers have demonstrated that choosing complex and unique passwords for many accounts is not feasible in general [30].

[31]This option differs from the local **auth/crypto** options we considered for SuperLock because the Hub has to act as an intermediary to other OCF devices, whereas one may want to isolate a SuperLock from the rest of the local network.

point of failure that leaks the credentials of every device relying on it if it's attacked. As a result, **expansive crime** and perhaps some **small crime** and **hacktivists** might additionally be willing to circumvent this security option.

5. **auth/biometric/voice**: the Hub could use people's voice signatures to authenticate the administrator. One would need to find a way to impersonate a user's voice or break the underlying implementation, as well as physical presence, to defeat this option. Assuming that OCFMakers would only deploy this option if it was fairly reliable, probably only **government** could ever carry out such attacks.

6. **auth/biometric/fingerprint**: the Hub could have a fingerprint scanner that authenticates administrators for the web app by their fingerprints. Since this attack would require physical presence as well as a way to lift fingerprints or break the underlying implementation, probably only **government** could circumvent this option.

There are several options to protect against **MITM onboarding**. These options, whose names begin with **ini**, include:

1. **ini/first**: on initial bootup, trust the first device to contact the Hub to be the administrator for onboarding, and set up a secure channel to continue onboarding from there. This would prevent just about any attacker from using **MITM onboarding**, unless the attacker has a persistent presence on the local OCF network and is able to make contact with the Hub before the legitimate administrator. Probably only **government** or maybe **APT** could do this consistently.

2. **ini/PIN**: on initial bootup, have the Hub emit an out-of-band random PIN through audio when it's being setup, which the administrator could enter on to whatever device is being used to facilitate the setup process.[32] This would

[32]The random PIN would be used as a pre-shared key for secure key exchange and subsequent encrypted communication.

prevent **MITM onboarding** unless the attacker could either record the audio and enter it on the MITM device faster than the administrator does.

3. **ini/button**: use a physical button on the Hub to start the initialization process for a brief period of time, and follow the procedure of **ini/first** from there. Like **ini/first**, this does depend on a race, but it is more difficult here, because race starts at a time controlled more directly by the administrator.

Finally, there are several options for dealing with the **command overload** means, whose names start with **install**:

1. **install/store**: have the administrator select and install appropriate commands via a store within the web application hosted by OCFMakers. To improve usability and prevent look-alike commands from being installed, when a device is connected to the Hub, it could provide a pointer to the correct commands. Beating this option would require getting by OCFMakers' certification process. While this could theoretically be fool-proof, the amount of spam in the iOS and Google Play stores leads us to believe that any attacker could occasionally fool some users.

2. **install/approve**: when a device or service tries to connect to a Hub, the Hub will audibly ask whether or not to accept the new endpoint's commands.[33] Cloud services, which are not installed on the local OCF network, could connect if the administrator allows them to do so through another administrative device. For instance, she could sign up for a cloud service on her phone, and, upon registration, the service could then send a request through the phone to the administrator's Hub. Similarly to **install/store**, any attacker could trick some users by using names that sound similar to the name of the legitimate device whose commands are being overridden.[34]

[33]If nobody is around at the moment that voice approval is asked for, the Hub could ask again later. It's not difficult to imagine that this could be implemented with negligible usability burden.

[34]One might imagine that, with this option, an attacker could spam a user by repeatedly asking her to accept new command patterns. While a naive implementation of this security option may allow this, there are simple additional mechanisms that could largely mitigate this. For instance,

3. **install/discovery**: have the administrator explicitly command the Hub to issue a request for new commands locally. The Hub would then verbally describe each new set of commands. If a malicious device tried to install its own commands on the Hub, the administrator could note the unexpected installation and remove it in the web application. As with **install/approve**, any attacker could gain some success with sound-alike names.

Table B.21 summarizes these options.

Table B.21: Security options for the Hub. Dollar signs indicate relatively how many resources needed to overcome a security option.

Option	Means						
	spoof admin	**admin commands**	**MITM onboarding**	**eavesdrop**	**modify traffic**	**spoof device**	**overload commands**
encryption[a]				$$$$			
auth/password/vendor	$	$					
auth/password/admin	$$	$$					
auth/crypto/local	$$$	$$$			$$$	$$$	
auth/crypto/remote	$$$[b]	$$$[b]			$$$[b]	$$$[b]	
auth/biometric/voice	$$$	$$$					
auth/biometric/fingerprint	$$$	$$$					
ini/first			$$$				
ini/PIN			$$$[c]				
ini/button			$$$[c]				
install/store							$$
install/approve							$$
install/discovery							$$

[a]Most other security options rely on this.
[b]Could be a central point of failure for all managed devices.
[c]Adds second factor over **ini/first**

Security vs. features

Whatever authentication options are chosen, only other OCF endpoints with compatible authentication schemes will be compatible with the Hub.

a Hub could simply reject repeat requests for command patterns that have already been rejected. There may be ways to get around this, but it would be difficult.

It's difficult to envision a way for remote cloud services to connect to a Hub using the **install/discovery** to prevent command overload.

Security vs. security

Table B.21 contains most of the information needed to see the security tradeoffs between the security options. Aside from this, one should note that most of these options can be chosen together. However, **ini/first** cannot be chosen with the other **ini** options, and only one of the **install** can be chosen.

Security vs. cost

encryption and the **auth/crypto** options cost a lot to develop in the first place, but it would not cost a lot for OCFMakers to incorporate existing crypto libraries into their code to take care of the most expensive aspects of these options.

Besides the base implementation costs, **auth/crypto/global** requires there to be a third party CMS to manage credentials, which is not a cheap thing for a third party to develop and support, but the marginal costs of the Hub's participation in it are minimal.

Both **auth/password** options would be relatively cheap to implement—password checking itself is easy to implement, and password input would only require minor modifications to the web application.

Relying on **auth/biometric/voice** for authentication would have high costs and pose a high risk, as its not clear whether such an authentication scheme could be developed to sufficient reliability at a cost affordable by OCFMakers.

Many devices already support **auth/biometric/fingerprint** authentication. OCF-Makers could probably choose to develop its own sensors and software in-house for higher development costs or purchase COTS fingerprint authentication components from a third party supplier. Either way, though, this option is extraordinarily expensive, as this option requires substantial hardware modifications that would raise the production costs of every Hub.

ini/first has minimal development and no per unit costs. **ini/PIN** would cost slightly yet manageably more to develop, but it can also be implemented without introducing new hardware capabilities. The **ini/button** would probably have development costs that are similar to those of **ini/PIN**, but it would impose higher hardware costs on each Hub produced. Like **auth/biometric/fingerprint**, the increased production costs of a hardware modification like this are substantial.

All **install** options impose small but nonzero costs on to other devices in order to be compatible with the Hub. It's worth noting, though, that it already takes some degree of third party investment to develop compatible command patterns in the first place.

install/store would require OCFMakers to invest a considerable amount in developing and maintaining the store; certifying that command patterns are legitimate would be particularly taxing. **install/approve** and **install/discovery**, by comparison, would likely have small development costs.

Security vs. usability

With the computational resources that Hubs were likely to have anyway, none of these security options will likely impose a noticeable performance cost.

The **auth/password** options take time to enter correctly and must be remembered, and each of these inconveniences grows with the complexity of the password. With complex and unique passwords, **auth/password/user** could easily become irrational or incentivize users to come up with workarounds to their own passwords [30, 41, 74].

The **auth/crypto** options require particular devices to be used to access the web app. This would be a pain if those privileged devices were lost, stolen, or damaged. Besides this constraint, though, the usability burden of **auth/crypto/local** and **auth/crypto/global** is minimal. In fact, they could improve usability overall by coordinating permissions between devices—some permissions could automatically and smartly be propagated to minimize how much the people using the devices actually have to micromanage them. In the case of **auth/crypto/local**, the Hub itself would do this management. With **auth/crypto/global**, the structure of the CMS itself, which is outside of OCFMakers' control, would determine the amount of coordination that could be automated. On the one hand, it could feasibly allow coordination between devices and services around the world, but, on the other, it may not facilitate automatic coordination at all.

Alternatively, the **auth/biometric** options prevent remote administration. Additionally, while they could work well with administration controlled by voice commands, they would be clunky and cumbersome to coordinate with the web app.

Biometric authentication can also be unreliable—if a Hub could not correctly recognize authorized **auth/biometric/voice** or **auth/biometric/fingerprints** almost every time, authentication would quickly become annoying. For **auth/biometric/voice** in particular, the developers would have to anticipate that people's voices can change based on their mood, age, and health.

The onboarding options—**ini/first**, **ini/PIN**, and **ini/button**—would all be pretty easy and convenient for most users to carry out. The latter two would require a bit of extra work, but it probably would not bother most people.

When connecting to new devices and adding new command patterns, **install/approve** puts the least burden on users, as long as precautions are put in place to mitigate spam. **install/discovery** takes hardly more effort, but it does require more positive action of the user compared to the reactive nature of **install/approve**. The **install/store** would take the most effort, but anyone who uses a smartphone app store could attest to, the usability burden here is still small. It may be worth noting that each these three options could actually improve usability over none of them at all, as they give clear feedback to an administrator that her Hub is set up to accept voice commands for a given device or service. Without them, she would have to check manually in the web app on her own.

B.3.5 Architectural Synthesis

Table B.22 shows the sections of the OCF specification and IoTivity documentation that provide guidance and requirements relevant to the security options and their tradeoffs. After considering these pieces of documentation, OCFMakers are left having to choose **encryption** as a security decision, choosing one **auth/crypto** option, one **ini** option, and optionally choosing either **auth/biometric** option and up to one **install** option.

As with the SuperLock, the lack of guidance about credential types, CMS design, and ACL management leave a lot of uncertainty in the tradeoffs.

Table B.22: Relevant documentation from the OCF specification [67] and IoTivity [46], and a description of their impact on the previously identified security options and their tradeoffs. All specification documents are from v1.1.0.

Documentation	Impact
Security 9.3.6	**auth/password** options not allowed
Security 9.3	**auth/biometric/voice** and **auth/biometric/fingerprint** not within OCF, but still acceptable as additional authentication factors
Security 9.3.3	Description of Asymmetric Authentication Key Credentials, suitable for **auth/crypto/local**, and **auth/crypto/remote**
Security 9.3.5	Description of Certificate Credentials, suitable for **auth/crypto/global**
Security 5	**encryption** and one kind of **auth/crypto** required
Security 11.2.3	Enumerates suitable ciphers for **encryption**, substantially reducing development costs
Security 5.1.1.2	Access Manager Service for **auth/crypto** boosts usability but decreases security (suitable for consideration as a sub-option)
Security 12.1	ACL guidance is currently TBD
Security 7.3.4	**ini/first** supported
Security 7.3.5	**ini/PIN** supported
Security 7.3.8	**ini/button** supported
IoTivity	standard software implementation of **ini/first** **ini/PIN**, cutting down software development costs
Core 11.3.2.2	**install/discovery** cheap to implement within OCF specification
Core 11.3.2.4	**install/approve** cheap to implement within OCF specification
Security 6	**auth/crypto** options would substantially mitigate **command overload**. Accordingly, **install** options become less critical

B.3.6 Tradeoff Resolution

From the Architectural Synthesis, we know that **encryption** must be chosen as a security decision. The OCF specification and IoTivity take care of most of the development work for this option [67].

First we consider which of the two **auth/crypto** options to choose. The latter will be cheaper if there is already a third party providing an external CMS to rely on; otherwise, building and supporting one is beyond the capabilities of OCFMakers. If one does exist and is widely used by many of the devices and services that OCFMakers hopes will interoperate with Hubs, then **auth/crypto/global** should be chosen for added value of compatibility with other devices.

Otherwise, if third party global CMS exists but is not the de facto authentication standard for OCF devices, the decision between these options falls on weighing cost, usability, and security. **auth/crypto/local** cannot be bribed and is not a single point of failure for all Hubs, so it could be slightly more secure than **auth/crypto/global**. More specifically, this additional security would make it more difficult for **government**, **APT**, and **expansive crime** to carry out **spoof device** and **modify traffic** at scale. Additionally, **auth/crypto/local** could position Hubs as standard credential managers of many home devices, increasing their overall value. On the other hand, **auth/crypto/local** is more expensive for OCFMakers to develop. If OCFMakers can afford this cost, they should choose **auth/crypto/local** for the added usability and security value.

Either way, choosing this option naturally allows these credentials to help authenticate administrators as well. OCF compliant devices with web browsers and administrator-level credentials, then, could simply access the web app by presenting their administrator credential. This process can be done rather securely and transparently to the user. The only security risk is that the credential is stolen from the administrative device or that the administrative device is not trustworthy (from the administrator's perspective.) In practice, this would likely mean that a homeowner's smartphone is compromised. While this can happen, many security measures are already put in place to these devices.

Biometrics could be used to offset this (relatively small) risk. The additional per unit cost and usability burden of **auth/biometric/fingerprint** cannot be justified; however, **auth/biometric/voice** could potentially be a justifiable second factor for authenticating to the web application. An administrator could simply be prompted to command her Hub to unlock the web app after trying to log in with her phone (or similarly authorized but potentially compromised device.) This would protect well against **spoof admin**, but it would not provide full protection against a compromised phone (which could send **admin commands**.)[35] However, the only attackers likely to be a threat at this level at all are **government** and **APT**, and they could likely find ways to circumvent **auth/biometric/voice** if needed, so this option is not worth its tradeoffs.

ini/PIN should be chosen over the other OTM options. It is more secure than **ini/first**, and it is already cheap because it is implemented in IoTivity and uses planned hardware capabilities, unlike **ini/button**. Most users will not be inconvenienced by the need to enter a PIN one time.

From the architectural synthesis, we already know that **auth/crypto/local** and **auth/crypto/global**—of which one will be chosen as a security decision—will largely prevent any random device or service from performing **command overload**. A malicious device or service could plausibly find a way to get a Hub to trust it enough to accept command patterns, but the credentials owned by the endpoints associated with overloaded commands could be used to discern between the two of them. With the ability to discern between them, a Hub could simply ask which endpoint the overloaded command is meant for.[36] If this becomes a burden for a user, the web app would presumably allow an administrator to block an unwanted and potentially malicious device or service.

Still, it would make sense to use **install/approve**. Knowing that a new device or service is set up with the Hub is a usability benefit that will probably outweigh its usability burdens. It also provides an extra layer of protection against **command overload** and costs almost nothing. **install/discovery** would not work by itself, as it does not allow users to connect a Hub with cloud services in any elegant manner.

[35]The compromised phone could just wait until a user unlocks the web app and take control after that.

[36]Even without malicious intent, a Hub will probably have to do this. For instance, the command "hail me a ride share to the mall" might provoke a Hub to ask "do you want a Lyft or an Uber?" As this consideration has nothing to do with OCF, this case study does not consider it in detail.

Likewise, the **install/store** would cost substantially more to develop and support, would require much more effort from administrators, and would have negligible security benefit over the other options.

Thus, the final security decisions at this stage of the framework are **encryption**, **ini/PIN**, **install/approve**, and either **auth/crypto/local** or **auth/crypto/global** for communicating with other devices and services as well as authenticating for the web app.

B.3.7 Discussion

This case study strengthened support for some of the observations made following the smart lock case study. Notably, following PRSD led the hypothetical OCFMakers developers through the process of identifying and thinking critically about the tradeoffs involved between various options in the Hub's security design. In coming to these decisions, the way that the processes in PRSD have the engineers explicitly link harms, attackers, and means, as well as link means and security options, made the analysis much easier and more efficient than trying to weigh options in an ad hoc manner. Moreover, every aspect of the analysis and their outputs contributed at least somewhat to the final security decisions, so there was no part of the analysis that was redundant or unnecessary. Finally, the architectural synthesis made it just as clear as before that the OCF specification needs more guidance on designing or choosing secure CMSs and ACLs.

Additionally, this case study, when compared to the previous one, demonstrated that different situations and threat models can call for different decisions between similar competing security options. With the smart lock, it made the most sense for each lock to generate its own local asymmetric cryptography keys for authentication. With the Hub, the case is much less clear—depending on relative costs, it may be preferable to rely on an external CMS. Similarly, while using an out-of-band PIN to bolster onboarding security was too expensive for SuperLocks, the extra costs for Hubs are nearly negligible, so the small extra security is worthwhile. In both of these cases, a more rigid framework prescribing one alternative or the other would not be perfect for these two case studies; it could guide one into a suboptimal decision.

This case study also gives an indication about how well PRSD scales to larger projects. Not only is the Hub substantially more complex than a smart lock, its interactions with other devices and services are more varied and uncertain, and its potential attack surface is much larger. Still, it was straightforward to decompose these complexities into a practicable threat model. Likewise, although the number of means and security options increased over the initial smart lock analysis, the analysis was still tractable. If development of these hypothetical devices were to continue into greater granularity, then the increased analytical demand of the Hub over the lock would probably compound. That said, since reapplications of PRSD over sub-options typically only considers relative competing sub-options, instead of all sub-options at once, there's no reason that the analytical *complexity* of using PRSD on the case studies should increase too much. In other words, while applying PRSD to the Hub throughout its development will inevitably take more time than it would to do the same to the smart lock, we predict that the difference in analytical tractability between the case studies would remain more or less constant throughout.

As a stray observation, it's interesting how drastically the Architectural Synthesis changed the tradeoffs at stake for mitigating **command overload**; it's just as interesting how easy these changes were to incorporate into the existing process outputs. When digging into the OCF architecture about resource discovery, it became clear that the leading authentication options (**auth/crypto/global** and **auth/crypto/local**) would automatically play a large role in preventing **command overload**. Previously, they were not considered relevant to this particular means. This changed the relative security benefits of the **install** options that were originally relevant to overload. Despite being such a large change, adjusting the tradeoffs accordingly was so simple that it justified only a few sentences of explanation between the Architectural Synthesis and Tradeoff Resolution analysis descriptions. In the end, these changes made it so that only one of the original three options (**install/approve**) was even worthy of consideration as a security decision. Even then, its ultimate inclusion as a security decision was only justified because the OCF architecture made its development and marginal costs nearly negligible.

B.4 Robot Vacuum

B.4.1 Given

In this case study, we will consider a hypothetical OCF-compatible household vacuum robot called “Vacubot” being developed by the equally hypothetical startup “V Corp.” This mobile robot will drive around, vacuuming up trash with maximum autonomy.[37] It can indicate when it is full with an LED on its top side. It has low-to-the-ground sensors (e.g., ultrasonic, light, and camera) for finding its way around and developing an internal model of the house layout. Under the hood, it uses AI to construct that model and determine how to vacuum it efficiently.

Let’s suppose that V Corp. is developing Vacubot in a world where OCF compliance is already ubiquitous. Vacubots have no direct controls, but will instead be controlled entirely through OCF commands by other appropriate OCF devices. We further assume that there standard data types that Vacubots adhere to in their exposed OCF resources, so other devices can automatically make sense of Vacubots and present relevant controls to users with whatever UI a device supports. For instance, an OCF remote control smartphone app may generate a custom set of controls from the OCF resources, whereas an OCFMakers Hub (see Section B.3) could determine relevant voice commands. Perhaps homes in this future will often have some yet-unimagined device that could automatically come up with and send convenient and efficient commands to Vacubots so that humans never have to manually operate it at all. Vacubots will rely on the onboarding process to determine what OCF endpoints to trust.

Vacubot conspicuously receives software updates over the air via a proprietary and explicitly non-OCF communications protocol. This is the only non-OCF communication it engages in.

[37]It can handle small stairs, but could never scale furniture.

Emptying the garbage collected by a Vacubot is an entirely manual process performed by humans.

It's not so quiet and small that it's sneaky, but it's still quiet enough that most people will never be annoying by the sounds it makes.

B.4.2 Situational Modeling

Capabilities

From the given information, we can specify several capabilities that Vacubots will have, which are listed in Table B.23.

Table B.23: Vacubot capabilities.

Capability	Description
move	move around the house
sense	sense objects near the ground
vacuum	vacuum dirt
communicate	send OCF communications over the local network and the Internet
update	receive Internet updates over a proprietary protocol
data	store sensor data, state, configuration, and models for vacuuming
AI	run AI algorithms that model house layout, efficient paths, and house dirtiness

Environment

Vacubots will be designed strictly for indoor use. They are mobile devices which takes commands from other arbitrary OCF devices. They could run into people, pets, and potentially valuable things, albeit with minimal force.

Stakeholders

Table B.24 details the relevant stakeholders, which follows the same structure and reasoning as the stakeholder analyses for SuperLocks (Tables B.2 and B.11) and Hubs (Table B.17.) Here, it is worth noting that "house objects" include any home object that a Vacubot could run into. "Peace of mind" refers to the peace that is ruined when one is annoyed; its inclusion will make more sense in the next section. It's also worth noting that we assume that V Corp. engineers will have below-average security expertise. Unlike Lockr's situation, V Corp. probably could not justify hiring security experts, and, unlike OCFMakers, V Corp. is not large enough to probably already have some, anyway.

Table B.24: Stakeholder Models

Type	Stakeholders	Interests and Assets	Expertise	Resources
Users	homeowners	Vacubot, OCF network, data, credentials, availability, house objects, peace of mind	minimal	minimal
Admins	homeowners	Vacubot, OCF network, data, credentials, availability, house objects, peace of mind	minimal	low
Developers	V Corp. engineers	company revenue	low[a]	full time job
Support	V Corp. engineers	company revenue	low[a]	full time job
Third parties	Vacubot-compatible product developers	company revenue, IoT products, data		
Public	People who use the Internet	Internet access, time		

[a]Relatively low security expertise compared to other developers

B.4.3 Threat Modeling

Harms

Multiple harms that could result directly from Vacubots, indirectly from Vacubots, and to Vacubots themselves. These harms are detailed in Table B.25.

Attackers

Table B.26 lists potential attackers who may want to carry out the harms enabled by Vacubots. Their expertise and resources mirror those described in the SuperLock

Table B.25: Potential harms resulting from Vacubots

Harm	Description	Assets at risk
leak data	sensitive data is lost to attackers	datai, company revenue
hit things	Vacubot runs into things	house objects, peace of mind, company revenue
troll	vacuuming places at inconvenient times	peace of mind, company revenue
unavailable	the Vacubot ignores commands	availability, company revenue
pivot harm	the Vacubot is exploited and used as a proxy to cause harm through other devices on the OCF network	OCF network, IoT products, data, company revenue
bot	Vacubot incorporated into botnet	Internet access, time, company revenue
pivot bots	the Vacubot is exploited and used as a proxy to incorporate other devices on the OCF network into botnets	Internet access, time, company revenue
credential theft	credentials for other accounts, devices, or services stolen	credentials, company revenue
replace	Vacubot is damaged and must be replaced	Vacubot, available, company revenue

(see Section B.1.3 and Section B.2.3) and Hub case studies (see Section B.3.3).

Table B.26: Potential attackers with an interest in Vacubot harms. Exclamation points after harms indicate relatively higher expected interest in causing that harm.

Attacker	Description	Harms of interest
prankster	people looking to cause mischief	**hit things**, **troll**, **unavailable**, **pivot harm**, **credential theft**, **replace**
hacktivists	activists who make statements through hacking	**leak data**, **hit things**, **troll**, **unavailable**, **pivot harm**, **bot**, **pivot bots**, **credential theft** (!), **replace**
burglars	people interested in burglarizing and stealing physical homeowner assets from a home	**leak data**, **pivot harm**, **credential theft** (!)
government	agents of law enforcement or espionage	**leak data**[a], **pivot harm**[a], **credential theft**[a]
APT	highly motivated, well-funded, expert, highly targeted, and patient attackers	**pivot harm**[b], **credential theft**[b]
expansive crime	organized crime looking to steal data or amass botnets	**bot** (!!), **pivot bots** (!!), **credential theft** (!!)

[a]APT will have an strong interest in causing this harm to a very small number of special individuals, and those targeted users will likely know the risks.

[b]States will have an strong interest in causing this harm to certain suspects or activists, but these cases are extremely rare, and those users will likely know the risks.

Means

Table B.27 details the means relevant to the OCF architecture through which attackers may cause the harms enabled by a Vacubot.

As with the SuperLock and Hub, **spoof admin** and **spoof user** differs from **admin commands** and **user commands** in that the former two exploit some weakness in identification and authentication and the latter two exploit some weakness in access control. In either case, we assume that administrator-level access could enable an attacker to get root on a Vacubot.[38]

[38]See Section B.1.3 for a justification of this assumption.

Table B.27: Means through which attackers could cause harms by leveraging Vacubot OCF communications.

Means	Description	Harms
spoof admin	impersonating an administrator	**leak data**, **hit things**, **troll**, **unavailable**, **pivot harm**, **bot**, **pivot bots**, **credential theft**, **replace**
admin commands	sending commands that entail administrator-level actions	**leak data**, **hit things**, **troll**, **unavailable**, **pivot harm**, **bot**, **pivot bots**, **credential theft**, **replace**
spoof user	impersonating an authorized user	**troll**, **unavailable**, **credential theft**
user commands	sending commands that entail user-level actions	**troll**, **unavailable**, **credential theft**
eavesdrop	listening in on communications between the Hub and another device	**credential theft**
modify traffic	actively modify OCF communications between a Hub and another endpoint	**leak data**, **hit things**, **troll**, **unavailable**[a], **pivot harm**, **bot**, **pivot bots**, **credential theft**, **replace**
MITM onboarding	man-in-the-middling the initial onboarding process	**leak data**, **hit things**, **troll**, **unavailable**, **pivot harm**, **bot**, **pivot bots**, **credential theft**, **replace**

[a]Compared to **spoof admin**, **admin commands**, and **MITM onboarding**, **unavailable** is of particular concern here.

B.4.4 Security Option Enumeration

Security options

This section presents a list of high-level security options that a company like V Corp. may actually consider. As with all other security designs, it is infeasible for V Corp. to think of, let alone fully consider, every potential option for mitigating threats to Vacubots. These options do, however, cover every means.

encryption: use a strong encryption scheme to make all OCF communications confidential. Most other security options depend on having this option. Additionally, a best-practices and correct implementation would prevent **eavesdrop** almost entirely and prevent **modify traffic** from causing any harm aside from **unavailable**.

There are several ways of authenticating identity that may work for Vacubot. For all of them, assume some accompanying permission scheme (such as ACLs) that is smartly implemented to control access to Vacubot resources.[39] These options each address **spoof admin**, **admin commands**, **spoof user**, and **user commands**. Additionally, they all rely on having **encryption**. The authentication options, whose names begin with **auth**, are:

1. **auth/passwords**: use one unprivileged password could be used to authorize operational OCF requests (day-to-day activities), and use a separate password to authorize all higher-privilege requests, including administrative configuration and arbitrary operations on OCF resources hosted by a Vacubot. This allows administrators to limit their trust of non-administrative users and their devices by only giving them the unprivileged password. This could stop most **pranksters** and **burglars**, but would only partially slow down the other attackers.[40]

2. **auth/crypto/local**: use cryptographically secure public key credentials to au-

[39]The details of this permission scheme would be sorted out later as sub-options to whatever authentication approach is chosen.

[40]Passwords can be guessed or socially engineered. Most people do not choose hard-to-guess passwords, even with most complexity rules [3]. Indeed, researchers have demonstrated that choosing complex and unique passwords for many accounts is not feasible in general [30].

thenticate administrators, other devices, and services, and store the credentials locally on Vacubots themselves. Those users, administrators, devices, and services would need to present valid credentials in order to have their commands accepted. However the permission scheme is ultimately set up, it would at least distinguish between privileged credentials, which could authorize changes to credentials and permissions, and unprivileged credentials, which could not. Aside from those trusted devices being compromised, defeating these credentials would probably require finding a weakness in the underlying cryptographic protocol (or exploiting a bad implementation, which is outside the scope of PRSD.) These techniques are typically only available to **APT** and **government**, but it is difficult to imagine that these attackers would Vacubots this way with any appreciable frequency.

3. **auth/crypto/remote**: this is similar to **auth/crypto/local**, except that Vacubots would trust some CMS with generating credentials and organizing the permissions scheme. To contrast, under **auth/crypto/local**, administrative agents can request permission changes, but the underlying permission system is still part of Vacubot's design and stored locally. The CMS could be housed within the local home OCF network or be a global CMS—Vacubots will trust whatever they are told when being initially set up. Likewise, this option provides the same protection as **auth/crypto/local**. The only caveat is that, if the CMS is compromised, it could be a single point of failure for all devices that rely on it. This single point of failure could be an attractive target to **expansive crime** if they can exploit it at scale.

4. **auth/fingerprint**: require a user's biometric fingerprint to authenticate and authorize and command that the Vacubot receives. Since V Corp. does not want to assume a particular kind of device for inputting commands, they would accept fingerprint data sent over OCF traffic from a device with a compatible fingerprint reader and also have a fingerprint scanner on each Vacubot itself to authorize commands from devices without compatible scanners. To defeat

it, an attacker would need to find a way to impersonate the fingerprint of an authorized user, which probably only **government** could carry out reliably.

5. **auth/PIN**: aside from the above, Vacubots could also require a second factor to authenticate administrators for particularly sensitive actions (like adding a new user.) This factor would have the Vacubot display an out-of-band random PIN on its top, which an administrator would then enter into an administrative device to authorize the action. This would require an attacker to either be physically present to view a PIN and enter it. Phishing would be tricky and likely require the attacker to compromise the administrative device being used to enter the PIN. Only **government** could carry out this attack reliably.

6. **no credentials**: finally, Vacubots could be configured not to store any authentication credentials locally at all. Regardless of means, this would make **credential theft** impossible. Information for verifying credentials like public keys and password hashes could still be stored. As a slight variation, **minimal credentials** would restrict stored credentials to only the absolute minimum needed for a Vacubot to identify itself.

Finally, there are several options with which to protect the initial setup of a Vacubot from **MITM onboarding**. All these options, whose names begin with **ini**, require **encryption**:

1. **ini/first**: on initial bootup, trust the first OCF device to contact the Vacubot to be the administrator for onboarding, and set up a secure channel to continue onboarding from there. This would prevent just about any attacker from using **MITM onboarding**, unless the attacker has a persistent presence on the local OCF network and is able to make contact with the lock before the legitimate administrator. Probably only **government** or maybe **APT** could do this consistently.

2. **ini/PIN**: on initial bootup, have the Vacubot display an out-of-band random PIN through a screen on its top side when it's being setup, which the admin-

istrator could enter on to whatever device is being used to facilitate the setup process.[41] This would prevent **MITM onboarding** unless the attacker could either record the audio and enter it on the MITM device faster than the administrator does.

Finally, we include, for the sake of comparison:

no OCF: do not make Vacubot communicate over OCF. Instead, use buttons on its body to operate it. Naturally, this prevents any of the means from being exercised whatsoever.

Table B.28 summarizes these options.

Table B.28: Security options for the SuperLock. Dollar signs indicate relatively how many resources needed to overcome a security option.

Option	Means						
	spoof admin	**admin commands**	**spoof user**	**user commands**	**eavesdrop**	**modify traffic**	**MITM onboarding**
encryption[a]					$$$$	$$$$[b]	
auth/passwords	$$	$$	$$	$$			
auth/crypto/local	$$$	$$$	$$$	$$$			
auth/crypto/remote	$$$[c]	$$$[c]	$$$[c]	$$$[c]			
auth/fingerprint	$$$	$$$	$$$	$$$			
auth/PIN	$$$[d]	$$$[d]	$$$[d]	$$$[d]			
no credentials	$[e]	$[e]	$[e]	$[e]	$[e]		$[e]
minimal credentials	$[e]	$[e]	$[e]	$[e]	$[e]		$[e]
ini/first							$$$
ini/PIN							$$$
no OCF	$$$$	$$$$	$$$$	$$$$	$$$$	$$$$	$$$$

[a]All **auth** and **ini** options require this.
[b]**unavailable** is unaffected.
[c]Depends on external CMS. Could be a central point of failure for all managed devices.
[d]Augments other **auth** options to this level.
[e]Invalidates credential theft, affects nothing else.

HI *Security vs. features*

[41]The random PIN would be used as a pre-shared key for secure key exchange and subsequent encrypted communication.

auth/passwords, **auth/fingerprint**, and **auth/PIN** limit the degree to which Vacubots can be automated. The first either requires a user to be near a trusted device that can take the password as input, or users and administrators can circumvent this by setting up devices with password managers (which, as we will see, has other tradeoffs.) **auth/fingerprint** is similar in that each operation a Vacubot performs requires the user to either have a device with a fingerprint scanner handy or to physically touch her Vacubot. Likewise, **auth/PIN** requires line-of-sight physical proximity to the Vacubot for every privileged operation it performs. These options would thus prevent Vacubots from running when their operators are, say, asleep or at work.

auth/crypto/remote requires there to be some compatible CMS available for a Vacubot to operate at all. In this hypothetical world of OCF ubiquity, this will not usually be a problem, but it is still a tradeoff worth noting.

no OCF would prevent Vacubots from having any of the OCF compatibility features in its original design.

security vs. security

Table B.28 contains most of the information needed to see the security tradeoffs between the security options.

Additionally, it should be noted that all **auth** options are compatible with **minimal credentials**, yet the **auth/crypto** options are not compatible with **no credentials**, as the Vacubot needs to store its own credential. **auth/fingerprint** might be; there is debate about how well a valid fingerprint can be determined from the information stored about a fingerprint signature [71].

The **auth/crypto** options are mutually exclusive. Of these two, the former avoids adding to a central point of failure. Otherwise, the **auth** options can be used in combination.

The **ini** options are mutually exclusive. **ini/PIN** provides better security, as it requires the attacker to have a method of reading the PIN.

no OCF is incompatible with all the other security options. It provides complete security against all the means in this framework.

Security vs. cost

Anything requiring cryptography, which includes **encryption** and anything that requires it, is expensive to design securely from the ground up. However, if OCF supports any existing standard libraries that implement cryptologically secure algorithms, then the development costs are small.

Besides the base implementation costs, which are comparable to **auth/crypto/local**, **auth/crypto/remote** requires there to be a third party that develops and supports a CMS for Vacubots to rely on. This is a huge undertaking. For normal Internet traffic, entire companies exist that just provide CMSs [17]. Certainly, as a startup, V Corp. does not have the resources to develop and support such a CMS on its own at this time. That said, if Vacubots are designed to work with prevalent existing CMSs, the marginal cost to those CMS providers would be minimal, if anything.

Both password options would be relatively cheap to implement—password transfer and checking itself is easy to implement (with widespread existing algorithms), and V Corp. could rely on existing user interfaces of OCF devices to take password input.

Similarly, many devices already support **auth/fingerprint** authentication. V Corp. could probably choose to develop its own sensors and software in-house for higher development costs or purchase COTS fingerprint authentication components from a third party supplier. Either way, though, this option is extraordinarily expensive, as this option requires substantial hardware modifications that would raise the production costs of every Vacubot.

ini/first has minimal development and no per unit costs, as it can be implemented in software. **ini/PIN** would cost more to develop, and it cannot be implemented without introducing new hardware capabilities that would substantially increase Vacubot production costs.

auth/PIN would have the same costs as **ini/PIN**. Further, with one PIN option, the marginal cost of adding in the other one is almost zero.

no credentials does not have an inherent cost of its own.

no OCF actually saves all the development costs of making Vacubots OCF-compatible. That said, it imposes the additional costs of designing a UI on the body of the Vacubot. This could be similarly priced or even more expensive than making Vacubots OCF-compatible, as V Corp. still needs to develop all the same features, but it has to do so from scratch instead of taking advantage of existing standards and the existing UIs of other OCF devices. Furthermore, **no OCF** dramatically increases the marginal costs of producing Vacubots because of the extra UI components it will need on its body, which, again, would already be in place on other devices if Vacubots were OCF-compatible. **no OCF** does not save on any marginal costs that might offset this increase.

Security vs. usability

With the computational resources that Vacubots are likely to have anyway, none of these security options will likely impose a noticeable performance cost. In this context, AI takes much more computational ability than **encryption**.

auth/passwords is highly unusable. Not only are passwords inherently unusable for a number of reasons [16], the extra burden of having to enter a password for every command to a Vacubot defeats much of the point of having an OCF-compatible automated vacuum cleaner to begin with. It still saves some work over a non-OCF vacuum, but not much. Saving the password or passwords on devices with password managers could greatly improve the usability of these options, although this is not something that V Corp. has any influence over.

auth/fingerprints take less time to input than passwords, but they still impose the extra burden of forcing users to take action to approve of everything that a Vacubot does. Furthermore, if a homeowner doesn't have a device with a fingerprint scanner, then she would have to physically touch the Vacubot to approve every action. Either way, like passwords, this defeats much of the point of having an OCF-compatible automated vacuum cleaner to begin with. It still saves some work over a non-OCF vacuum, but not much.

auth/PIN has the same low usability as **auth/fingerprint**, except it always requires one to be within line of sight of the Vacubot. This is not as bad as having to touch it, but it is less convenient than using a remote fingerprint scanner.

By contrast, the **auth/crypto** options, if designed and implemented well, would be largely unnoticed over no authentication. The only difference is that homeowners would have to use authorized devices to interact with the Vacubot. This is not a huge burden; if one doesn't have one's phone, then being unable to vacuum is a trifle compared to the other resulting inconveniences.

Indeed, **auth/crypto/remote** could actually make the Vacubot even more usable than it would be otherwise by streamlining its permission management with other OCF devices it manages. This could save homeowners a lot of hassle and even potentially automatically set up some coordinated behavior between Vacubots and other devices. Automated coordination is still possible with **auth/crypto/local**, but only if all the other devices are set up to deal with the kind of credential that Vacubots generate automatically.

The onboarding options—**ini/first** and **ini/PIN**—would each be pretty easy and convenient for most users to carry out. The latter would require a bit of extra work, but it probably would not bother most people to do input one PIN when they buy a new device.

no OCF is the least usable at all, as it blocks all the convenience of remote and automatic operation that OCF compatibility offers. These conveniences would be the Vacubot's competitive advantage, so this is a significant sacrifice to make.

B.4.5 Architectural Synthesis

Table B.29 covers the sections of the OCF specification and IoTivity that provide guidance or requirements relevant to the security options and their tradeoffs.

To summarize, the specification requires **encryption** to be used, and it does not allow either **auth/passwords** or **auth/fingerprint** to be used. Exactly one **auth/crypto** option must be used, although there is little guidance for choosing between or implementing either one. Either **ini/first** or **ini/PIN** can be chosen and developed at low cost (although the production costs are unchanged), and some guidance is given for choosing between them. **auth/PIN** can also be chosen and developed at low cost, although no guidance for it is given. **Minimal credentials** is also a viable option. As an alternative to all of this, **no OCF** can be chosen.

Table B.29: Relevant documentation from the OCF specification [67] and IoTivity [46], and a description of their impact on the previously identified security options and their tradeoffs. All specification documents are from v1.1.0.

Documentation	Impact
Security 9.3.6	**auth/passwords** not allowed
Security 9.3	**auth/fingerprint** not allowed
Security 9.3.3	Description of Asymmetric Authentication Key Credentials, suitable for **auth/crypto/local**, and **auth/crypto/remote**
Security 9.3.5	Description of Certificate Credentials, suitable for **auth/crypto/remote**
Security 5	**encryption** and one kind of **auth/crypto** required
Security 11.2.3	Enumerates suitable ciphers for **encryption**, substantially reducing development costs
Security 5.1.1.2	Access Manager Service for **auth/crypto** boosts usability but decreases security (suitable for consideration as a sub-option)
Security 12.1	ACL guidance is currently TBD
Security 7.3.4	**ini/first** supported
Security 7.3.5	**ini/PIN** supported
IoTivity	standard software implementation of **ini/first** **ini/PIN**, cutting down software development costs. Similarly decreases development costs of **auth/PIN**.

B.4.6 Tradeoff Resolution

To reiterate, if Vacubots are to be OCF compatible, then **encryption** needs to be chosen.

The tradeoffs between **auth/crypto/local** and **auth/crypto/remote** are complicated. From a cost perspective, the latter is only a viable option if V Corp. can reasonably expect that a sufficient number of consumers have access to an adequate third party CMS to use; the startup cannot afford to make and support such a CMS itself. Even if both are available options, the tradeoffs remain complicated. **auth/crypto/local** entails the costs of designing some sort of credential and permission scheme, and **auth/crypto/remote** entails the costs building interfaces to the existing third party CMSs that Vacubots are to be compatible with. This trade-off could fall either way, although if there is a small set of standard CMSs with well-documented interfaces,[42] **auth/crypto/remote** would be cheaper to develop and implement. Further complicating this decision, the usability of **auth/crypto/remote** is likely to be higher than that of **auth/crypto/local** if the CMS can streamline the ACLs associated with the Vacubot. Thus, if viable, **auth/crypto/remote** is more likely to be slightly preferable to **auth/crypto/local** in terms of cost and usability (although there are a number of variables to check for both of these.)

That said, **auth/crypto/remote** is likely less secure than **auth/crypto/local**, because it contributes towards a central point of failure for all OCF devices that depend on it. Furthermore, some CMSs can be bribed or otherwise compromised. While both options would be somewhat susceptible to attacks from **government**, these weaknesses might make **auth/crypto/remote** additionally susceptible to **expansive crime**.[43] That said, if **expansive crime** can attack CMSs at scale, Vacubots are unlikely to sway this attacker's decision making or contribute significantly to the botnet that these attackers would build, anyway. Thus, while **auth/crypto/local** does technically have some security advantages over **auth/crypto/remote**, those advantages are incredibly unlikely to matter when translated to the actual harms that could result. The developers should simply choose the more usable and cost-effective of **auth/crypto/local** and **auth/crypto/remote**.

The other security options are less complicated to choose from. **ini/first** is preferable to **ini/PIN**, because it saves on the marginal production price of Vacubots. While this makes **MITM onboarding** less secure, it is difficult to think of a situation where that would make a significant difference. Even with **ini/first** alone, an attacker needs to have already compromised the local OCF network. While **MITM onboarding** could technically enable any potential harm, an attacker who has al-

[42]Although one would expect such standards in a future of ubiquitous OCF devices, the OCF specification does not provide any help in CMS and ACL design.

[43]While **APT** might also have the resources, there is no harm that would make this avenue of attack worthwhile to it. Even if there are some incredibly critical devices or services that choose to trust Vacubots and thereby make **pivot harms** particularly harmful, those critical devices and services probably have a poor security design with lots of other vulnerabilities, anyway. As a result, **APT** could just target them directly.

ready compromised the network could likely carry out most of them, anyway. She couldn't necessarily carry out **leak data**, **hit things**, **troll**, and **replace**, but an attacker with persistence likely would not be interested in these things.

auth/PIN is even less worthwhile, because it hurts usability more than **ini/PIN**. Thus, it will also be discarded.

If Vacubots are to be OCF compatible, **minimal credentials** should be chosen. It minimizes the harm possible through **credential theft** and comes with essentially no tradeoffs.

The final decision is whether or not OCF compatibility is worthwhile. With **encryption**, **minimal credentials**, **ini/first**, and either **auth/crypto/local** or **auth/crypto/remote**, many of the harms are significantly mitigated. The only realistic ways to to carry out any of the means is to either compromise other trusted devices on the network and maintain persistence or, under **auth/crypto/remote**, compromise the CMS. In all these cases, there are hardly any situations where there would be an attacker with enough interest in the potential harms to put in the effort to carry out any of the means. Even then, the marginal harm that could result from attacking a Vacubot is hardly more than what the attacker could probably pull off anyway.[44]

Furthermore, the OCF standards make the development costs of these options low, and they all have minimal effect on features and usability. **no OCF**, on the other hand, would greatly diminish features and usability and raise the marginal cost of Vacubots substantially. With no outstanding security threat in the way, the choice to reject **no OCF** is easy. Thus, **encryption**, **minimal credentials**, **ini/first**, and either **auth/crypto/local** or **auth/crypto/remote** are the final security decisions.

[44]The only exception would be if some particularly sensitive device chooses to trust Vacubots enough to make *pivot harms* a major threat. However, V Corp. cannot do anything to prevent other devices from trusting Vacubots, and it is not their responsibility if other devices have poor security designs.

B.4.7 Discussion

This case study provides further evidence for some of the observations made in the previous case studies. Again, following PRSD led the hypothetical developers through the process of identifying and thinking critically about the tradeoffs involved between various options in the Hub's security design. In coming to these decisions, the structure of PRSD made the analysis easier and more efficient than trying to weigh options in an ad hoc manner. Additionally, as before, there were no redundant or unnecessary aspects of the structure of PRSD. Finally, the lack of specification or guidance of CMS design continued to make it difficult to estimate and compare tradeoffs between options that rely on them.

The only security decision in common between the three case studies is encryption, which is strictly required by the OCF.[45] Thus, as hypothesized, varying tradeoffs do lead to different security options being reasonable. Moreover, PRSD helps identify and grapple with these tradeoffs; a less systematic framework would make this analysis more difficult, and a more rigid framework would lead to suboptimal security designs in some or all of these case studies.

One might feel that this case study has a lot of analysis for a simple, low-risk device. Especially when it turns out that some of the stronger security options were ultimately rejected (or at least not accepted outright, in the case of **auth/crypto/local**), it may seem that PRSD was a waste of time on Vacubot. However, by doing this analysis, V Corp. would have concrete evidence to demonstrate why they made the decisions they did. If Vacubots are ever successfully attacked, this evidence could potentially help V Corp. win a legal liability case.[46] Additionally, this analysis potentially saves V Corp. a lot of money if they would have otherwise chosen more expensive security options like **ini/PIN**. It would also save them money if they would have chosen less secure options and if Vacubots were subsequently hacked. Thus, the analyses in this case study could be well worth their cost in the long run.

[45]They all have some key-based authentication, but the chosen design and tradeoffs between contending designs.

[46]See Section 5.6 for more information on this possibility.

In addition to previously identified gaps in the OCF specification, it would have helped if the OCF provided an option for Vacubot to assert limits to how much other devices trust it. More specifically, it would be nice if Vacubots could claim that they are only trustworthy for reporting details of its own status and that it should never command other devices to do one thing or another. If it could do this, then **pivot harms** and **pivot bots** would have been non-factors. As it is, there is no way to anticipate for sure how much other devices will trust Vacubots. In the spirit of the Principle of Least Privilege [72], other devices should not trust Vacubots much, and they certainly should not trust them beyond their self-reported status. However, there is nothing that enforces, let alone guarantees, this trust minimization. Thus, it is difficult to estimate the risk of **pivot harms** and **pivot bots**. While it could be argued that other devices' trust in Vacubots is not the responsibility of V Corp., the uncertainty and risk are still there. Having a "do not trust me" aspect to OCF would make this much cleaner.

Despite these shortcomings, OCF compliance is still worthwhile in this scenario. This case study explicitly compared the advantages and disadvantages of OCF compliance with the tradeoffs of foregoing OCF functionality entirely, and, despite some slightly heightened security risks, OCF compatibility was overwhelmingly worthwhile. Not only did OCF compliance make Vacubots more valuable due to increased interoperability, there are ways that standardization cut costs: standard implementations cut down development costs, and V Corp. is able to use the user interfaces of other existing products instead of having to design, implement, and produce its own. Thus, with sufficient adoption, there are times that the Open Connectivity Foundation's standards might be worthwhile in the end.