Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things

Tianlong Yu†, Vyas Sekar†, Srinivasan Seshan†, Yuvraj Agarwal†, Chenren Xu‡
†Carnegie Mellon University, ‡CECA Peking University

ABSTRACT

The Internet-of-Things (IoT) has quickly moved from the realm of hype to reality with estimates of over 25 billion devices deployed by 2020. While IoT has huge potential for societal impact, it comes with a number of key security challenges—IoT devices can become the entry points into critical infrastructures and can be exploited to leak sensitive information. Traditional host-centric security solutions in today's IT ecosystems (e.g., antivirus, software patches) are fundamentally at odds with the realities of IoT (e.g., poor vendor security practices and constrained hardware). We argue that the network will have to play a critical role in securing IoT deployments. However, the scale, diversity, cyberphysical coupling, and cross-device use cases inherent to IoT require us to rethink network security along three key dimensions: (1) abstractions for security policies; (2) mechanisms to learn attack and normal profiles; and (3) dynamic and context-aware enforcement capabilities. Our goal in this paper is to highlight these challenges and sketch a roadmap to avoid this impending security disaster.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: *General*-security and protection; C.2.1 [**Computer-Communication Networks**]: *Network Architecture and Design*-distributed networks

General Terms

Security, Design

1 Introduction

The Internet-of-Things (IoT) has quickly moved from hype to reality; Gartner, Inc. estimates that the number of deployed IoT devices will grow from 5 Billion in 2015 to 25 Billion in 2020 [4]. Like other disruptive technologies, such as smartphones and cloud computing, IoT holds the potential for societal scale impact by transforming many industries as well as our daily lives.

HotNets '15 November 16–17 2015, Philadelphia, PA USA

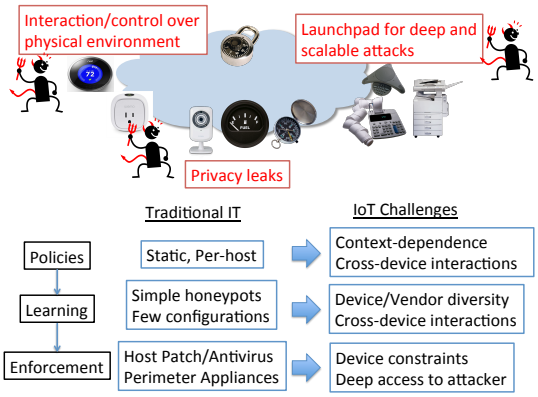

Figure 1: IoT security challenges and how/why conventional IT security approaches are found wanting

While IoT has huge potential, it also comes with its share of challenges. Most vendors only deal with parts of the IoT ecosystem and, typically, their priorities have been providing novel functionality, getting their products to market soon, and making them easy to use. Unfortunately, security and privacy risks have not received as much attention. Since IoT devices will typically be embedded deep inside networks, they are attractive attack targets and may become the "weakest link" for breaking into a secure IT infrastructure [17], or for leaking sensitive information about users and their behaviors [18]. These are not hypothetical concerns as several actual attacks have already been reported. For example, IoT devices were used as bots to launch DDoS or spam [3], smart meters were hacked to lower utility bills [15], and handheld scanners were compromised to enter logistics firms [6].

Today's IT security ecosystem, which relies on a combination of static perimeter network defenses (e.g., firewalls and intrusion detection/prevention systems), ubiquitous use of end-host based defenses (e.g, antivirus), and software patches from vendors (e.g., Patch Tuesday), is fundamentally ill-equipped to handle IoT deployments (Figure 1). Specifically, the scale, heterogeneity, use cases, and device and vendor constraints of IoT means that traditional approaches fall short along three key dimensions:

- **Types of policies:** IoT devices can interact with other devices via explicit channels (e.g. a single app may use multiple IoT devices) or implicitly by affecting the *physical* world around them (e.g., an IoT light bulb may trigger an IoT light sensor). Thus, compromised IoT devices can affect both applications that use them explicitly as well as applications with implicit or indirect *cross-device de-*

Row	Device	Device Num.	Vulnerability
1.	Avtech Cam	130k	exposed account/password
2.	TV Set-top box	61k	exposed access
3.	Smart Refrigerator	146	exposed access
4.	CCTV Cam	30k (by IP)	unprotected RSA key pairs
5.	Traffic Light	219	no credentials
6.	Belkin Wemo	>500k (estimated)	open DNS resolver, use for DDoS
7.	Belkin Wemo	>500k (estimated)	exposed access, bypass app

Table 1: Examples of Known IoT Vulnerabilities.

pendencies. The result is that the security for an IoT deployment are likely to be complex and dynamic since they depend on both physical (e.g. environmental parameters) and computational (e.g., the state of other related devices) contexts.

- **Learning signatures and anomalous behaviors:** The *diversity* of IoT devices and vendors inevitably means that traditional approaches of discovering attack signatures (e.g., honeypots) will be insufficient and/or non-scalable. Furthermore, there might be implicit dependencies where device interactions are indirectly coupled through the environment. Thus, we need new mechanisms to learn signatures and infer such cross-device dependencies to inform security policies.
- **Enforcement mechanisms:** Since IoT devices operate deep inside the network, traditional perimeter defenses are ineffective. At the same time, IoT devices typically do not run full-fledged operating systems, require low-power consumption and are resource constrained. Moreover, the longevity of these devices means that vulnerable devices (e.g., default passwords, unpatched bugs) remain deployed long after vendors cease to produce or support them. Thus, traditional host- or device-centric mechanisms (e.g., antivirus, patches) are impractical to expect in an IoT world. Finally, given that the environment and device behaviors can change rapidly, we need to rapidly reassess and update the system's security posture. Unfortunately, today's security enforcement schemes stem from a static mindset and cannot handle such dynamics.

Rather than chase the hopeless goal of IoT devices that are secure-by-construction, we need pragmatic "bolt-on" solutions for securing IoT that acknowledge the realities of marketplace (e.g., existing devices, poor software practices, patch unavailability) and the practical challenges associated with IoT (e.g., resource or software constraints). Unlike traditional IT ecosystems where host-based detection and prevention are prevalent, we believe that the device and ecosystem limitations of IoT will need the the network to (re)emerge as the key vantage point for enforcing security policies. In the rest of the paper, we elaborate on the challenges with respect to policies, policy learning, and enforcement and discuss preliminary ideas for rethinking network security for IoT.

2 Motivation and Overview

To highlight some of the security challenges in IoT, we first present a number of reported IoT vulnerabilities, and the number of devices affected, from known public databases. Next, we describe the challenges with cross IoT device interactions. Finally, we present the issues in using current enforcement mechanisms and highlight the requirements for IoT security.

Device	Cross-device policies	Typical Example
NEST Protect	188	If Nest Protect detects smoke, then turn Philips hue lights on.
Wemo Plugin	227	Turn of WeMo Insight if SmartThing shows no body is at home.
Scout Alarm	63	Activate your Manythings Camera if Alarm is Triggered.

Table 2: Cross device policy examples.

2.1 Motivating Scenarios

IoT Vulnerability Cases: Table 1 lists a small subset of IoT device vulnerabilities found from SHODAN [14] and other sources. The examples come from different types of IoT devices, including: cameras (Row 1 and 4), Set-top box (Row 2), Smart Appliance (Row 3), Traffic light (Row 5) and Smart Plugs (Row 6 and 7). The first three cases are examples where the devices have hardcoded default username/passwords ("admin/admin" for Avtech cameras) or devices with open IPs, ports and protocols that can be accessed (Row 2 and 3). The fourth example is a CCTV setup with unprotected RSA key pairs in the firmware image for ≈30K devices [20]. Here, an attacker can gain access to the devices and mount more complex attacks on the internal networks, turn on/off devices, or compromise the privacy of individuals. To date, these vulnerabilities remain unpatched. The traffic light vulnerability (Row 5) allows unfettered access of 219 traffic lights, enabling an attacker to change traffic lights and even cause accidents [14]. The last two examples (Row 6,7) are vulnerabilities in the popular Belkin Wemo line of smart home products. The Wemo devices run a open DNS resolver which was used to mount a DDoS attack. The second vulnerability allows open access to the devices across the Internet (not just on the LAN), which means that their data (power usage) be accessed and they can even be turned ON/OFF remotely. In summary, these anecdotes suggest that vulnerable IoT devices can create a significant threat to the security of our networked infrastructures and compromise the privacy of individuals.

Cross-device dependencies: Like typical network devices, IoT devices can communicate *explicitly* with each other. For example, a networked thermostat (e.g., NEST) can control the air-conditioning system in a smart home. However, unlike traditional devices, IoT devices can also be coupled through the physical environment leading to *implicit* dependencies. For instance, a temperature sensor can be connected to a service like IF-This-Then-That (IFTTT) [7] to open windows to cool down a space when the air-conditioning is not active. Thus, an attacker could compromise the smart plug (e.g., Belkin Wemo) to turn off the air-conditioner in a room and trigger a temperature increase, which would, in turn, cause the the windows to open and create a physical security breach. Such cross-device dependencies are quite common. Table 2 shows the number of cross-device dependencies [7] and typical cases for three widely used IoT devices: NEST Protect [9], Wemo Insight

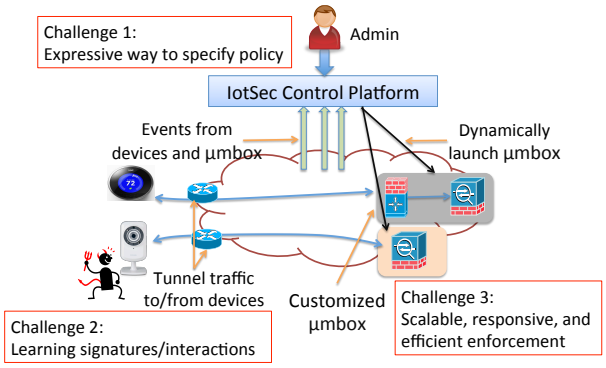

Figure 2: High-level vision of IoTSec

Switch [1] and Scout Alarm [13].[1] To mitigate potential attacks, we need to reason about normal behaviors, and, thus, need systematic mechanisms to discover and express such dependencies.

Broken Enforcement Mechanisms: Traditional enforcement mechanisms are unlikely to be effective in IoT deployments for a number of reasons. First, there are no host-based defenses (e.g., antivirus) solutions due to resource constraints on these devices and the lack of a common programming environment or operating systems. For example, even antivirus systems for embedded systems, such as Commtouch Antivirus [2], require 128 MB RAM, while most IoT devices use single-thread microcontroller (8051, MSP430, ATMEL series) with $\leq$ 2 MB RAM. Second, unlike traditional IT devices, IoT devices lack effective automated software updates. The current process of patching IoT vulnerabilities is via manual firmware updates, and that too per device/vendor. Unfortunately, due to the longevity of IoT devices, software updates will likely be unavailable (e.g., vendor may not support updates or no longer exist) or be too late to prevent early exploits. Third, existing network security mechanisms largely stem from a static perimeter-defense mindset (e.g., IDS and firewall at the gateways). With vulnerable IoT devices embedded deep inside networks and dynamic behaviors that change with operating context, such classical approaches quickly become ineffective.

2.2 System Overview

From the examples in the previous section, we can generalize two key observations for securing IoT: (1) host-based approaches are ineffective and we need to depend on network-based solutions, since IoT devices contain significant numbers of unpatched vulnerabilities and have limited resources; (2) traditional static perimeter defenses are unable to secure IoT devices, since these devices are deployed deep inside the network, with their physical and computational context constantly changing. To enforce security policies based on the dynamic context, we envision a new *software-defined approach to IoT security*, where we can: (a) rapidly develop and deploy novel network defenses tailored to IoT use cases and (b) dynamically customize the network's security posture to the current operating context of different devices and the environment.

Figure 2 shows a high-level vision of our IoT security architecture called IoTSec. While our approach and ideas are quite general, we focus on residential and commercial IoT deployments.[2] IoTSec envisions customized $\mu mboxes$ (micro network-security functions) that act as security gateways for each IoT device. A logically centralized IoTSec controller monitors the contexts of different devices and the operating environment and generates a global view for cross-device policy enforcement. Based on this view, it instantiates and configures individual $\mu mboxes$ and the necessary forwarding mechanisms to route packets to these $\mu mboxes$. This vision is quite general and can naturally support a range of IoT management models; e.g., directly connected devices vs. IoT hubs [12] vs. smartphone-controlled [7]. To enable immediate deployment, we assume the enterprise has a well-provisioned on-premise cluster with a pool of commodity server machines. In a home scenario, we envision an upgraded version of an IoT router (e.g., Google OnHub [5]) with compute capabilities. Each IoT device's first-hop edge router or wireless access point (AP) is configured to *tunnel* packets to/from the device to the cluster or an IoT router.

Given this high level vision, there are three key outstanding challenges: (1) An expressive way to specify the types of policies to enforce (Section 3); (2) How can we learn signatures of malicious attacks and patterns of normal (cross-device) behavior to inform these policies (Section 4); and (3) How to implement scalable, responsive, and efficient enforcement mechanisms (Section 5).

3 Policy Abstractions

In this section, we discuss why existing policy abstractions, e.g. firewall rules or IoT management protocols, are not expressive enough to handle the types of security and safety properties for the scenarios described in Section 2. Then, we present an expressive-but-inefficient finite state machine (FSM) policy abstraction that captures key environmental, cross-device, and security contexts. We end with open challenges.

3.1 Strawman solutions

To motivate the problem, consider two natural strawman solutions, one each from the traditional (IT) network and the IoT domains. In traditional IT security, a simple policy abstraction used by firewalls and IDSes, is a set of $Match \rightarrow Action$ pairs, where the $Match$ predicate is typically specified in terms of packet headers (L3-L4 ACLs) or payloads (e.g., IDS). More advanced policies also include *connection state* $State, Match \rightarrow Action$; e.g., a stateful firewall allows incoming traffic if an outgoing connection was established earlier. These abstractions do not work in the IoT context because: (a) the security-relevant behavior

[1] NEST Protect is a smoke and carbon monoxide alarm. Wemo Insight Switch is a smart plug that monitors energy usage. Scout is a next-generation home alarm.

[2] Other settings (e.g. drones, automotive IoT, autonomous vehicles, process control systems) are outside the deployment scope we consider here as they entail different connectivity infrastructures.

for IoT depends on *environmental context* which is missing (e.g., a thermostat controlling the HVAC system is normal if the user is present and anomalous otherwise); and (b) there are potential *cross-device* interactions that impact security (e.g., policy for the smart oven depends on the state of the fire alarm).

In the IoT domain, IF-This-Then-That (IFTTT) [7] is a popular abstraction, supporting recipes such as "If smoke emergency, set lights to red color" or "If Sighthound detects a person at home when I'm away set light to red color" [7]. While these capture cross-device interactions, they have three fundamental security limitations. First, they do not capture the security-relevant context of devices (e.g., are these unpatched). Second, they assume recipes are independent, which can either lead to conflicts or safety violations. For instance, in our example both the smoke alarm and the Sighthound rules could be active simultaneously leading to ambiguity. Third, it is tedious for users to reason about possible device interactions and their effects, leading to incomplete specifications that can be exploited by an attacker.

3.2 Proposed approach

To address the limitations of the strawman solutions, we now sketch an expressive albeit "brute force" solution to capture the relevant environmental context, security-relevant context, and cross-device interactions. First, suppose we have $\mathcal{D}$ networked IoT devices, and each $D_i \in \mathcal{D}$ has a security context C_i, which can take one or more values (e.g., "normal" or "suspicious" or "unpatched"). Second, suppose we have $\mathcal{E}$ environmental variables (e.g., temperature, smoke, window), and each variable $E_j \in \mathcal{E}$ can take one or more discrete values (e.g., Temperature=High/Low, Window=Open/Closed, Smoke=Yes/No). Now, we can represent the set of possible *states* $\mathcal{S}$ of the system in terms of these device contexts and environmental variables. In the limiting case, the total number of states is combinatorial; i.e., $|\mathcal{S}| = \prod_{i,j} |C_i| \times |E_j|$.

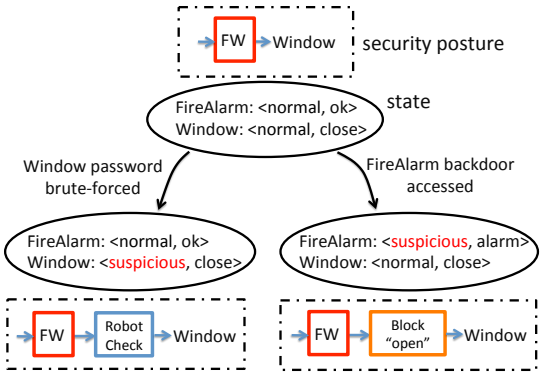

Figure 3: Policy abstraction illustration for IoT network with fire alarm and window actuator.

Given these preliminaries, we can define our security abstraction as follows. For each state $S_k \in \mathcal{S}$, we define the *security posture* for each device $Posture_{S_k,D_i}$. This security posture specifies the set of security modules through which the traffic for the device needs to be subjected (e.g., "proxy"-ing capabilities) as well as the set of anomaly detection and signature detection rules that need to be applied in this specific case. By construction, this policy abstraction is expressive and can capture the necessary environmental context and security-relevant context of the device, as well as cross-device interactions.

To see this policy abstraction in action, let us revisit our example in Figure 3, and formally specify the policy for each state. For instance, when the FireAlarm's backdoor is accessed[3], the state becomes $\mathcal{S} = \{C_{FireAlarm} = suspicious, E_{FireAlarm}, C_{Window}, E_{Window}\}$, then the policy enforcement is to block any "open" message sent to the window actuator to stop potential break in.

While this is an expressive abstraction, there are two obvious open questions. First, with respect to the enforcement logic (Section 5), this brute-force enumeration may not be practical as the number of devices and states scale. Second, the state explosion makes it difficult to check for potential policy conflicts or correctness issues [33]. We believe that in practice it might be possible to prune and collapse this giant FSM by exploiting some domain-specific opportunities. For example, if we know that two specific device types are inherently independent, or if the intended security posture is the same for a set of similar states, then we can potentially prune the state space.

4 Learning Security Policies

At a high level, security detection and prevention systems rely on two standard approaches: (1) detecting the *signatures of the attack* and (2) *detecting anomalous behaviors* that deviate from normal activity.

In case of IoT, learning signatures using simple honeypot-like mechanisms will not scale with the diversity of devices and deployments — we would need several thousand honeypots to ensure coverage for every specific device "SKU" as opposed to vendor or class of device (e.g., Google Nest version XYZ rather than "thermostat"). One potential avenue is to extend prior work for decloaking environment-specific malware that can support multiple modes of execution [23]. However, we expect the scale and diversity of IoT to be much higher than the space of possible browser environments, which these prior efforts targeted. Similarly, applying simple anomaly detection to IoT also does not scale since the range of possible normal behaviors is large and potentially very dynamic and taking cross device interactions is further challenging. We discuss below a preliminary sketch of solutions to each of these issues.

4.1 Learning signatures

To address the diversity challenge, we envision a *crowd-sourced* repository that allows users who have deployed a specific IoT device SKU to share attack signatures of interest that they have observed with other users who have deployed the same SKU. The repository would offer a simple publish-

[3]A suspicious event indicating an attacker is trying to compromise the FireAlarm to open the window for break in.

subscribe interface, where users could publish traces or signatures, expressed in a common format, which other users could subscribe to.

There are several technical challenges to address to make this crowdsourcing work: (1) *Incentivizing reporting:* Enterprises are often reluctant to acknowledge breaches for fear of monetary impacts, or due to bad public relations, or for the fear of being hacked; (2) *Privacy:* Sharing information raises concerns about the potential for accidentally leaking private information; and (3) *Data Quality:* With any crowdsourcing solution, there is the risk of noisy data (accidental or adversarial) which may inadvertently lead to a denial of service (e.g., if a malicious or misconfigured signature blocks all traffic).

While these are valid concerns, we believe that we can build upon prior work to address them. For instance, to address incentives (1), we envision an *anonymous* publish-subscribe system [21] with a feature that give priority notifications to users who contribute signature data. To address the privacy challenge (2), we can institute anonymization routines or use other types of privacy-preserving techniques to limit information leakage [28, 29]. Finally, to address the data quality challenge we can borrow techniques from the crowdsourcing literature; e.g., use reputation or voting mechanisms [31, 37] to deal with incorrect reporting.

4.2 Learning cross-device interactions

While crowdsourcing could work for individual devices, it is not practical to uncover combinations of device interactions, especially as these interactions are coupled to the specific operating environment. Since each deployment is likely to be customized differently, covering all possible interaction modalities by simply observing the behaviors in the wild is unlikely to work.

To address this concern, our key insight is that we can abstract the environment and different classes of IoT devices, and use these abstract models to systematically reason about the space of possible interactions. To this end, we envision building a *library* containing abstract models of different *classes of devices* (e.g., toaster, microwave, smart bulb rather than specific instances) that capture key input-output behaviors and interactions with environment variables. In this respect, we can build upon prior work in modeling cyberphysical systems as simple FSMs that have been traditionally used for verification and vulnerability assessment [32].

We assume that there will be a broader community effort to develop and refine such models. One potential approach to build these abstract model of devices and their effect on the environment is to observe deeply instrumented (controlled) IoT testbeds with instances of different classes of devices (e.g. a toaster, a bulb). Then using a combination of actually actuating devices into different states and observing their effects on the environment, we could build an empirical model of devices. Automatically extracting these model specifications is an interesting direction for future work.

Given the abstract model, we can apply techniques like fuzzing [19, 22] or "monkeying" [8, 10, 34]. Specifically, we can think of the states of each IoT device model and the environment as potential input variables for fuzzing. Then, we run multiple fuzz tests to explore the space of possible behaviors. We expect that device interactions will likely be *sparse* as it is constrained by the specific environment that couple interactions; e.g., physical proximity and network topology. Thus, fuzzing can give us reasonable coverage over the space of acceptable behaviors.

As a next step, such models can also be used to automatically identify potential multi-stage attacks due to cross-device interactions; e.g., triggering device X to transition to state S_X and then using that to reach an eventual goal state (e.g., unlocking the door). To this end, we can borrow ideas from attack graph analysis in the security literature [30, 36].

5 IoTSec Enforcement

Now, we turn to the challenges in practical enforcement. As discussed earlier, host-based protections will no longer be effective and we need network-level mechanisms to provide: 1) context-based enforcement according to system state S_k; 2) agile enforcement to change $Posture_{S_k,D_i}$ implementation according to the constantly changing system state S_k.

Next, we discuss the potential of extending SDN and NFV to secure IoT devices, identifying key challenges. In terms of the control plane, the key challenge is *scale* and *responsiveness* in maintaining an up-to-date view of the global system state S_k and responding to environment changes. For the data plane, the key challenge is in developing efficient and flexible platforms for developing IoT defenses.

5.1 Control plane

We foresee two challenges in extending SDN mechanisms to IoT settings. First, traditional mechanisms for scaling SDN typically exploit the weak consistency semantics [24] needed in network management; e.g., critical network information like topology that need stronger consistency semantics typically do not change often. However, this is unlikely to be the case for IoT since changes in critical state of S_k that must be handled in a consistent fashion does change often. Second, we envision much more frequent reconfiguration of policies relative to existing SDN efforts. One possible approach to handle the consistency and update challenges is to logically partition the set of IoT devices depending on the frequency in the interaction dependencies. Thus, we can have a hierarchical control architecture where frequently interacting components are handled together by a low-level controller and infrequent interactions are handled at the global controller.

5.2 Data plane

Unlike traditional IT deployments with a single firewall/IDS for the enterprise, we envision many micro-middleboxes ($\mu mboxes$), each can be customized for a specific device type and can be rapidly instantiated and frequently recon-

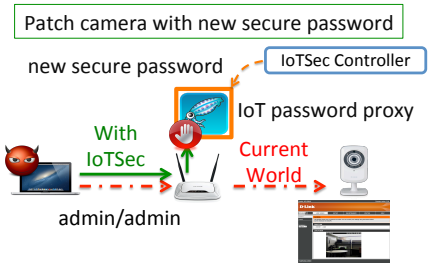

Figure 4: Patching exposed password for a home camera.

figured when the environment changes. This introduces a different set of challenges with respect to (1) resource management and (2) programming abstractions relative to concurrent efforts in the NFV space. To address (1), we can exploit the fact that the actual computation that each micro-middlebox performs will be lightweight and not need to operate at high traffic rates. Thus, we can create custom micro VMs [27] that can be rapidly booted/rebooted [26]. To address (2), we envision a lightweight Click [27] version akin to TinyOS [25] that can serve as an extensible programming platform for developing these micro-middleboxes. In addition, unlike many current middleboxes, the $\mu mboxes$ must support frequent reconfigurations without impacting the availability of IoT devices and services.

5.3 Proof of Concept

As a preliminary proof-of-concept, we have developed an early prototype of IoTSec. For the control plane, we use `OpenDaylight` [11] and for the data plane we use modified versions of `Squid` [16] and `Snort` [35].

Next, we discuss two use cases that show the early promise of IoTSec: 1) Providing security gateways for insecure devices and 2) Enforcing policies for cross-device interactions.

IoT security gateway: As discussed in Section 2, a common vulnerability is that IoT devices can ship with default admin passwords that allow adversaries to hijack the device and/or extract private data. In Figure 4, we use a D-link surveillance camera D-link surveillance camera which ships with a hardcoded admin password that the user has no interface to delete.[4] As shown by the red lines in the figure, any attacker can access the camera's management interface and images. To address this, we use a $\mu mbox$ (Ubuntu VM with a customized Squid proxy) to serve as a gateway that interposes on all traffic to the camera. By interposing on traffic, the $\mu mbox$ can enforce the use of a new administrator-chosen password to access the camera's management interface.

Cross-device policy: Next, we consider a simple two-device setting with a Belkin Wemo device and the D-link camera. In Figure 5, we consider a scenario where the remote attacker compromised a Wemo through a backdoor (no credential needed). Suppose this Wemo controls power source for a smart oven that is a potential fire hazard. Our

[4]Unsurprisingly: "admin/admin"

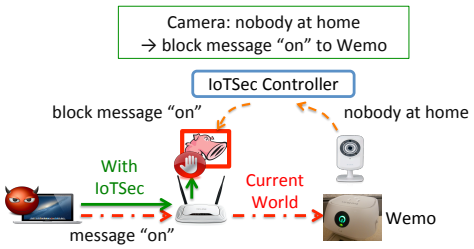

Figure 5: Enforce cross device policy.

cross-device policy encodes a common policy from IFTTT: ensure that the oven can be turned on only if the camera detects a person in the room. IoTSec enforces this policy using a $\mu mbox$ that accesses the status of the camera and uses `Snort` to interpose on traffic to the Wemo. Our $\mu mbox$'s policy is set to allow the "ON" messages to be sent to Wemo only if the global state identifies a person in the room and, thus, can prevent a remote attacker from causing damage via the Wemo vulnerability.

6 Conclusions

Whether we like it or not, the world is heading towards the catastrophic consequences of having a network with billions of insecure IoT devices with potentially unfixable flaws. We argue that traditional approaches to security are fundamentally at odds with the IoT ecosystem (e.g., perimeter defense, antivirus, patching) and inadequate to capture the dynamic environment and cross-device interactions that are common to IoT. Our goal in this paper was to articulate these challenges and chart out a roadmap to potentially promising solutions. While we have many unsolved questions, we hope that this paper acts as a catalyst to spark the debate on how to tackle this impending security disaster.

Acknowledgments

This work was supported in part by NSF award number CNS-1440056 and by Intel Labs University Research Office.

7 References

[1] Belkin Wemo. `http://www.belkin.com/us/Products/home-automation/c/wemo-home-automation/`.
[2] Commtouch Antivirus for Embedded OS Datasheet. `http://www.commtouch.com/uploads/pdf/Commtouch-Antivirus-for-Embedded-OS-Datasheet.pdf`.
[3] Fridge sends spam emails as attack hits smart gadgets. `http://www.bbc.com/news/technology-25780908`.
[4] Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015. `http://www.gartner.com/newsroom/id/2905717`.
[5] Google ON hub. `https://on.google.com/hub/`.
[6] Hackers attack shipping and logistics firms using malware laden handheld scanners. `http://www.securityweek.com/hackers-attack-shipping-and-logistics-firms-using-malware-laden-handheld-scanners`.
[7] IFTTT Recipes. `https://ifttt.com/recipes`.
[8] Monkey. `http://developer.android.com/tools/help/monkey.html`.
[9] NEST. `https://nest.com/`.
[10] Netflix Simian Army. `https://github.com/Netflix/SimianArmy`.
[11] OpenDayLight. `http://www.opendaylight.org/`.
[12] Samsung Smartthings. `http://www.smartthings.com/`.
[13] Scout Alarm. `https://www.scoutalarm.com/`.
[14] SHODAN. `https://www.shodan.io/`.

[15] Smart meters can be hacked to cut power bills. http://www.bbc.com/news/technology-29643276.

[16] Squid. http://www.squid-cache.org/.

[17] The Internet of Things Is Wildly Insecure - And Often Unpatchable. http://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-of-things-and-thats-a-huge-problem/.

[18] Will giving the internet eyes and ears mean the end of privacy? http://www.theguardian.com/technology/2013/may/16/internet-of-things-privacy-google.

[19] S. K. Cha, M. Woo, and D. Brumley. Program-adaptive mutational fuzzing. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 725–741, May 2015.

[20] A. Costin, J. Zaddach, A. Francillon, D. Balzarotti, and S. Antipolis. A large-scale analysis of the security of embedded firmwares. In *USENIX Security Symposium*, 2014.

[21] A. K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/subscribe in p2p networks. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 8–pp. IEEE, 2003.

[22] P. Godefroid, M. Y. Levin, and D. Molnar. Sage: whitebox fuzzing for security testing. *Queue*, 10(1):20, 2012.

[23] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert. Rozzle: De-cloaking internet malware. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 443–457. IEEE, 2012.

[24] T. Koponen et al. Onix: A Distributed Control Platform for Large-scale Production Network. In *Proc. OSDI*, 2010.

[25] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. A. Brewer, and D. E. Culler. The emergence of networking abstractions and techniques in tinyos. In *NSDI*, volume 4, pages 1–1, 2004.

[26] A. Madhavapeddy, T. Leonard, M. Skjegstad, T. Gazagnaire, D. Sheets, D. Scott, R. Mortier, A. Chaudhry, B. Singh, J. Ludlam, et al. Jitsu: Just-in-time summoning of unikernels. In *12th USENIX Symposium on Networked System Design and Implementation*, 2015.

[27] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. ClickOS and the art of network function virtualization. In *Proc. NSDI*, 2014.

[28] F. McSherry and R. Mahajan. Differentially-private network trace analysis. *ACM SIGCOMM Computer Communication Review*, 41(4):123–134, 2011.

[29] P. Mittal, V. Paxson, R. Sommer, and M. Winterrowd. Securing mediated trace access using black-box permutation analysis. In *HotNets*. Citeseer, 2009.

[30] X. Ou, S. Govindavajhala, and A. W. Appel. Mulval: A logic-based network security analyzer. In *USENIX security*, 2005.

[31] J. Pang, B. Greenstein, M. Kaminsky, D. McCoy, and S. Seshan. Wifi-reports: Improving wireless network selection with collaboration. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, MobiSys '09, pages 123–136, New York, NY, USA, 2009. ACM.

[32] A. Platzer. Verification of cyberphysical transportation systems. *Intelligent Systems, IEEE*, 24(4):10–13, 2009.

[33] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang. Pga: Using graphs to express and automatically reconcile network policies. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 29–42. ACM, 2015.

[34] L. Ravindranath, S. Nath, J. Padhye, and H. Balakrishnan. Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 190–203. ACM, 2014.

[35] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[36] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.

[37] K. Walsh and E. G. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *USENIX NSDI*, volume 6, 2006.