Hestia: Simple Least Privilege Network Policies for Smart Homes

Sanket Goutam
North Carolina State University
sgoutam@ncsu.edu

William Enck
North Carolina State University
whenck@ncsu.edu

Bradley Reaves
North Carolina State University
bgreaves@ncsu.edu

ABSTRACT

The long-awaited smart home revolution has arrived, and with it comes the challenge of managing dozens of potentially vulnerable network devices by average users. While research has developed techniques to fingerprint these devices, and even provide for sophisticated network access control models, such techniques are too complex for end users to manage, require sophisticated systems or unavailable public device descriptions, and proposed network policies have not been tested against real device behaviors. As a result, none of these solutions are available to users today.

In this paper, we present Hestia, a mechanism to enforce simple-but-effective network isolation policies. Hestia segments the network into just two device categories: controllers (e.g., Smart Hubs) and non-controllers (e.g., motion sensors and smart lightbulbs). The key insight (validated with a large IoT dataset) is that non-controllers only connect to cloud endpoints and controller devices, and practically never to each other over IP networks. This means that non-controllers can be isolated from each other without preventing functionality. Perhaps more importantly, smart home owners need only specify which devices are controllers. We develop a prototype and show negligible performance overhead resulting from the increased isolation. Hestia drastically improves smart home security without complex, unwieldy policies or lengthy learning of device behaviors.

KEYWORDS

IoT & network security; smart home; least privilege policy

ACM Reference Format:
Sanket Goutam, William Enck, and Bradley Reaves. 2019. Hestia: Simple Least Privilege Network Policies for Smart Homes. In *12th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '19), May 15–17, 2019, Miami, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3317549.3323413

1 INTRODUCTION

Internet connected home devices, once a niche product category, are now a normal part of consumers' lives. In the United States alone, 33.2% of homes have at least one smart device, and this number is expected to grow to 53.9% in the next four years [21]. The value of these devices comes in part from their ability to be automated and controlled in tandem from single interfaces like smart hubs. Not only do these devices provide the convenience of remotely monitoring temperature and carbon monoxide sensors, accessing video surveillance, tracking pets' movements, and remotely locking doors, they permit recipes like scheduling lights, music, and coffee for breakfast time. As a result, devices of varying computational ability manufactured by different vendors are filling homes, in some cases tripling the number of Internet-connected devices.

These devices, individually and in toto, create significant security concerns. Not only are they deployed to perform sensitive tasks (e.g., cameras monitoring children) but also play important safety roles (e.g., door locks, smoke detection). Moreover, repeated security flaws leading to severe operational failure [10], mass exploitation [3], and use of devices as a pivot to attack other networked devices [22] mean trust in the average device is largely misplaced. Popular devices have been found to be exploitable from the local network [8, 16], creating the very real threat that a single compromised device may then attack other local devices. Prior incidents [8, 22] where IoT devices have been exploited to compromise other systems on the same network make this threat clear. The fact that these devices are collections of diverse hardware and software complicates any and every host-based security solution. The sheer number of devices alone currently or soon-to-be deployed means that users would be overwhelmed with managing any host-based security solution.

In this paper, we present a new system, Hestia, whose goal is to reduce risk of compromise in smart homes by implementing a least-privilege network policy. We first note that smart home networks consist of a few controller devices, like automation hubs and personal assistants that provide user interfaces, and the remaining non-controller devices that perform sensing, monitoring, or actuating duties. We find that almost without exception, non-controller devices only communicate with controllers and the cloud, and never directly from non-controller device-to-device. The key insight behind Hestia is that we can isolate individual smart home non-controller devices to only communicate with controllers and the cloud and no other devices. This provides a drastic reduction in possible communication paths — approximating least-privilege access to the network for non-controller devices. Configuration is then incredibly straightforward: users need only specify which devices are controllers.

This paper makes the following contributions:

- We demonstrate that our controller–non-controller dichotomy is robust using a large public dataset of over 40 smart home devices [2].
- We then construct a prototype of Hestia as a SDN application for Open vSwitch; in so doing, we address policy management, enforcement, and device discovery.
- We demonstrate that Hestia provides significantly enhanced isolation with minimal performance impacts compared to a stock wireless access point.

We are not the first to explore network policies to constrain IoT device behavior. However, prior work either proposes enforcement mechanisms without testing or validating policies [15], or requires fine-grained policies [4] that require significant investment to specify or discover using (fallible) learning methods. This work provides a validated policy that can be deployed *today* with minimal effort.

2 IS A SIMPLE MODEL PRACTICAL?

In this paper we are interested in applying a least privilege network policy to reduce the risks posed by vulnerable IoT devices to other devices on the local network. The critical challenge in a least privilege policy is: how do we determine what privileges a device needs? While industry mechanisms like MUD specification [14] can provide fine grained device behavior descriptions, they are currently not implemented for today's devices, and may never be implemented for low-cost offerings. Other proposed systems [5, 11, 15] leverage machine learning to learn device behaviors, but still ultimately result in complex, unwieldy policies.

We hypothesized a simpler solution could be equally effective. We begin with the hypothesis that IoT devices primarily connect only to a smart hub or *controllers*, and not to other IoT devices or user devices (e.g., laptops). Loosely, controller devices receive control inputs from users, like Google Home or Alexa hubs, and execute certain actions on other non-controller IoT devices, like door locks or light bulbs. If this hypothesis held, we realized that a least privilege policy need only allow data flows between individual IoT devices and controllers, preventing compromised IoT devices from successfully targeting other non-controller devices. Such a policy can be implemented simply, with far less complexity or user input than other methods. In this section, we demonstrate using a previously-published large IoT dataset [2] that this approach is feasible, leading the way to the development of our system, *Hestia*. The following section then provides an insight on the goals of such a design followed by an overview of our system.

2.1 Policy Evaluation

In prior work, Alrawi et al. [2] released the YourThings dataset that includes 46 labelled smart home devices and the network traffic they generated during a manual assessment of each device. They recorded the device interactions on the network at 5 minute intervals over a period of 10 days. We use these captures to validate our hypothesis that a mere distinction of controllers and non-controllers is sufficient to achieve least privilege network policy.

2.1.1 Experiment. To test our hypothesis, we first manually classified the smart home devices in the YourThings data set into controllers and non-controllers. Our classification criteria was that devices which have one or more user-facing control interface (voice, app, etc.) and can execute actions on other IoT devices are controllers, while any other IoT devices are not. For instance, if a smart TV has Alexa integrated and the manufacturer's website details that it can be used to control other devices in the network then we identify it to be a controller. Whereas a smart light bulb may be accompanied by an app on the users smart phone, but in no scenario can it send instructions to any other IoT device in the network. In most cases, necessary information about the device was accessible directly from the manufacturers landing page for their device. This process was fast and could be repeated by users with limited technical expertise.

Table 1: Device categorization on the YourThings data set

Distinguishing Feature	No. of Devices	Category
None	26	non-controllers
Voice Assistant	10	controllers
Remote Control Hub	9	controllers
Home Router	1	controllers

Table 1 shows the number of devices that were classified as controllers and non-controllers in the data set and the distinguishing features used to determine this classification. We identified a total of 20 out of the 46 labelled devices to be a controller in the network.

The next step was to filter the recorded network data for local network communication and evaluate our policy. We used the `scapy` Python library to filter all the local network packets from each network capture and aggregated all device communications for each day of the recordings. Once we had the aggregated network communication data, we created a *src - dst* mapping of devices where a unique mapping was created on a per-day basis.[1] The goal was to achieve an insight on which devices are able to send messages to each other (within each 24-hr period), and thus if we found at least one instance of a packet exchange between two sets of devices on the network then we recorded it in our mapping.

2.1.2 Findings. Using the *src - dst* mapping of device interactions obtained for each day, we found that there were a total of 426 device-to-device communications that took place over the 10 day period. Since we created these mappings on a day-to-day basis, these include repeated instances of device-to-device interactions over several days. We determined that 54.69% of these device interactions were between devices that we had identified to be acting as the controllers in the network. We also found that 45.07% of connections occurred between controllers and non-controllers preserving our hypothesis that non-controllers need controllers to function. All of these packets conformed to our initial hypothesis that communication from IoT devices is constrained to controllers within the local network.

Our analysis revealed a single exception to the policy. We found that two packets were exchanged between two non-controllers: a D-Link DCS5009L camera and a Belkin Netcam. This was a UPnP device discovery initiated by the D-Link camera requesting the device details of Belkin Netcam. This device discovery was spurious and was not necessary for legitimate functionality. In fact, it serves as an example of the unnecessary and unauthorized network traffic we seek to prevent. While we believe this traffic was innocuous, it is similar to known attacks against the Belkin Netcam [17].

2.1.3 Takeaway. These empirical results validate our hypothesis that a simple network policy limiting IoT device traffic on the local network to only controller devices can be deployed in practice today with virtually no disruption to the proper function of these devices.

3 PROBLEM

As demonstrated in Section 2, non-controller devices only require network communication with the Internet and controller devices. This observation offers an opportunity to enforce a network access

[1]The devices and configurations vary from day-to-day in the YourThings dataset

control policy that is both simple to specify and approaches least privilege. In this section, we describe the problem via an example scenario. We then define the threat model for Hestia.

Problem Scenario: Setting up a seemingly innocent user appliance, like a smart coffee maker, involves three key steps. First, the user installs the coffee maker in the desired location and uses an accompanying smartphone app to connect to it, often via Bluetooth. Second, the coffee maker attaches to the home WiFi network. It either automatically uses the same WiFi SSID as the smartphone, or discovers all available WiFi SSIDs and asks the user, through the accompanying app, to select one and provide credentials. Finally, as the coffee maker connects to the WiFi network, remaining connected until changes are made by the user. Note that users are sometimes recommended to use network segmentation, or a separate SSID just for smart home devices; however, doing so complicates setup (e.g., if the SSID of the smartphone is cloned) and often adds network management overhead for average users.

The purpose of a smart coffee maker is quite specific: it brews coffee based on a schedule or when it receives a command from the user. It needs to be connected to the home WiFi network, and receive these user commands only from a supported controller platform. The coffee maker is a non-controller device and thus should not communicate with other non-controllers, like the printer, the refrigerator, or the security camera on the network. By having full access to the LAN, the coffee maker is thus over-privileged. Because these smart home appliances are essentially fully-featured, Internet-connected Linux computers, vulnerabilities in their implementation [8] mean attackers can use it as a pivot to attack the entire network. This paper seeks to define an approximation of least privilege network access for smart home devices that would mitigate such a network compromise, while balancing usability.

Threat Model: We consider a smart home network where users have multiple smart home appliances, like a smart coffee maker, all connected to a consumer-grade router. By deploying all devices on a shared network space, users put the same level of trust on all connected devices. We assume that an adversary can discover the vulnerable devices in a users' household using malicious smartphone apps [20], by exploiting HTML error messages [1], or by simply gaining physical control of the device [7]. Once the attacker has gained access to a device, they can maliciously gather sensitive information from other connected devices (e.g., unique device identifiers) [1], use lateral movement to steal user data from the network-attached storage unit [22], or discover a vulnerable security camera on the local network and attack it remotely [10]. Network compromises such as these are known as "Rube Goldberg" attacks, where attackers penetrate the local network through a vulnerable device, and then use it to exploit other connected systems. Note that many user devices, including NAS, coffee makers, and security cameras, are non-controller devices and do not need to interact with each other. Like other network access control systems (e.g. firewalls), we recognize that detection of attacks (e.g. NAT hole punching) and compromised devices (e.g. botnet infections [3]) will have orthogonal solutions [9, 15].

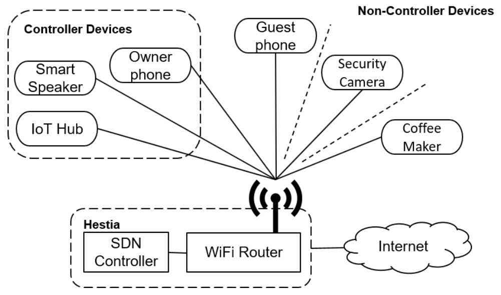


Figure 1: Hestia provides virtual partitions around non-controller devices while allowing controller devices to communicate with all non-controller devices and each other. All devices can communicate with the Internet.

4 HESTIA

As motivated by our empirical evaluation in Section 2, Hestia is designed to enforce a simple policy: *non-controller devices may only communicate with controllers and the WAN.* This policy allows for a simple policy specification, only requiring knowledge of the link-layer addresses of controller devices. While not explicitly included or evaluated in our design, we reasonably assume the existence of a mechanism for an end user to designate a device as a controller, either *a priori* or during first the connection. Enforcing this simple policy requires overcoming the following research challenges.

- *Existing LAN network access control mechanisms do not mediate between devices on the same LAN.* VLANs and separate WiFi SSIDs unnecessarily complicate network setup, and are incompatible with many consumer smart home devices that (a) automatically use the SSID of the smartphone performing setup, and (b) use multicast discovery protocols.
- *Multicast discovery packets from non-controller devices must not reach other non-controller devices.* Discovery protocols such as mDNS and SSDP commonly use multicast packets. Network layer multicast packets are broadcast to the link layer, allowing devices to respond if they are configured for that multicast address.

Hestia addresses the first research challenge using Software Defined Networking (SDN) primitives. Specifically, we modify an OpenWRT firmware to include an OvS (Open vSwitch) soft switch that relays packets to a Ryu-based SDN controller application (potentially running on the router). Through this SDN-based instrumentation, Hestia is able to mediate network communication between devices on the same LAN, without the need for VLANs or multiple SSIDs. To address the second challenge, Hestia provides a generic solution that supports mDNS, SSDP, and other multicast discovery protocols without specific knowledge of the protocols. To do so, Hestia uses the group table feature of OpenFlow 1.3 to selectively duplicate multicast packets at the link-layer and send a unicast version to controller devices.

Figure 1 overviews the Hestia architecture. The WiFi router instrumented with Hestia is configured with the MAC addresses of all controller devices. By default, a device is a non-controller device. Since non-controller devices can communicate with controller devices and the WAN, most devices will work automatically when they are connected to the LAN. When the first packet of a new flow arrives at the router's WiFi interface, the Open vSwitch sends the packet header information to the Ryu-based SDN controller application. The SDN controller inspects the source and destination MAC addresses. If the flow is allowed, an OpenFlow flow-mod is sent back to Open vSwitch, permitting subsequent packets in the flow to forward without the involvement of the SDN controller. If the destination MAC address is one of a set of multicast addresses used by device discovery protocols, Hestia uses the group table feature of OpenFlow 1.3 to define action buckets that automatically duplicate the payload and unicast send it to controller devices. The remainder of this section describes the design of Hestia.

4.1 Enforcement

Hestia is configured with a list of identified controller devices. Our implementation uses a JSON file enumerating the MAC addresses. We envision two ways for Hestia to obtain this list. A straightforward way is for the user to identify each controller device when it is added to the WiFi network for the first time. This specification could occur via the router's configuration web interface, or via a companion smartphone application. As an automated alternative, Hestia could be combined with IoT device fingerprinting system such as IoT Sentinel [15] (though "controller" annotations would be needed). Even with automation, Hestia should allow the end users to manually designate controller devices as needed (e.g., to add specific non-IoT devices such as smartphones and laptops). Note that not every smartphone or laptop needs to be a controller device. Only those requiring connections to other devices on the LAN need to be controllers.

Hestia uses Open vSwitch to mediate packets between all devices on the LAN. In normal WiFi access points (APs), the wireless interface behaves as a bridge between the wireless clients. If a frame originating from a wireless client is destined for another client connected to the same wireless interface, then the frame is directly sent to the other client without going through the network stack (bypassing Open vSwitch). To force frames to go through the network stack, we enable wireless isolation mode (also called AP isolation or peer-to-peer blocking), which is available in standard APs. By default, wireless isolation will prevent all wireless clients from communicating with one another. However, SDN can selectively re-enable communication that adheres to the policy [12]. To do so, we configure Open vSwitch to forward packets to the ingress port (`OFPActionOutput(ofproto.OFPP_IN_PORT)`) when communication is between two wireless clients.

We implemented the Hestia SDN controller application using Ryu. Ryu is relatively lightweight and Python-based, allowing it to run on newer commodity routers with more resources, or via a directly connected device (e.g., Raspberry Pi). Hestia installs flow rules based on the device categorization to create a categorical entity abstraction, as shown in Figure 1. Devices are classified as "non-controller" by default. All non-controller devices are isolated such that they can initiate connections to the Internet and controller devices, but not to other non-controller devices. Therefore, a controller device not designated as such will still have partial functionality (e.g., Internet connectivity). Once a device is specified as a controller, it may act as a traditional device on a LAN. Finally, whenever the controller list is updated, Hestia flushes all OpenFlow rules to ensure proper operation. In effect, this enforcement maps to a loose approximation of least privilege based on the network functionality, as we observed in Section 2.

4.2 Selective Device Discovery

Since most homes use dynamic IP assignment (i.e., DHCP) and do not run internal DNS servers, controller devices use discovery protocols to connect to smart home devices. For example, mDNS and SSDP discover devices by sending multicast packets, which reach the NIC of all devices on the LAN. Devices configured as mDNS or SSDP listeners consume the multicast packet and respond using protocol-specific conventions. Hestia carefully controls device discovery packets to prevent non-controller devices from performing network reconnaissance and discovery protocol-based attacks.

Hestia seeks a generic approach for handling device discovery that works without protocol-specific knowledge of mDNS and SSDP. Standard network implementations handle multicast packets by copying the original packet from the source and broadcasting it to all connected devices with a MAC layer destination address of the desired multicast group. To achieve selective isolation for multicast communication, Hestia must control which devices can receive the multicast packet.

To efficiently mediate multicast device discovery protocols, Hestia uses the *group table* feature in OpenFlow 1.3. Group tables allow an SDN controller to define action buckets that automatically duplicate the packet for each action target. When a non-controller sends a multicast packet, *Hestia* creates a group table that matches on the source MAC address of the non-controller and the destination MAC address of each multicast group. We then use the action buckets to specify an action for each controller MAC address. In doing so, Open vSwitch effectively converts multicast packets into unicast packets, without sending the packet payload to the SDN controller application. Note that multicast packets from controller devices are forwarded as normal.

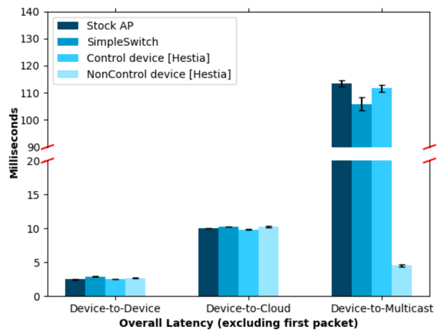
4.3 Implementation

We implemented Hestia on top of a custom OpenWrt firmware with Open vSwitch and Hostapd pre-compiled. We removed the default Linux bridge and separately added an Open vSwitch bridge. We used the Ryu SDN framework and implemented the Hestia SDN controller in 320 lines of Python.

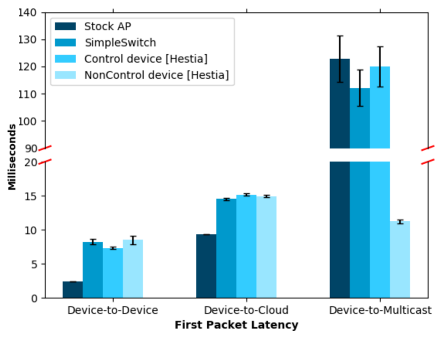
5 EVALUATION

The previous section described the design of Hestia. In this section, we show that Hestia has negligible performance impact. Our network evaluation explores a total 12 different experimental conditions. To evaluate the network performance of Hestia, we want to investigate the impacts on latency and throughput to all communication types: device-to-device, device-to-cloud, and multicast messages. For all message types, we also consider the difference in non-controller-to-controller flows and controller-to-controller flows. For each measurement, we compare Hestia to the stock OpenWRT firmware and the default "Simple Switch" Ryu app.

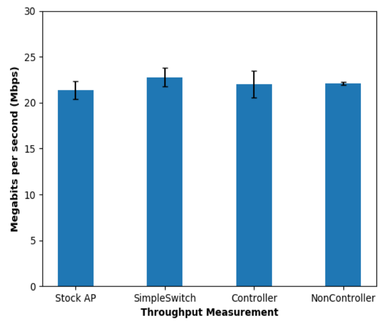
We measure three key variables: first packet latency, average (non-first) packet latency, and average throughput. We distinguish the first packet of each flow from subsequent packets because SDN



(a)



(b)



(c)

Figure 2: *Hestia* adds negligible overhead to latency or throughput when compared against a default SDN app

systems must forward the first packet of a new flow to the SDN controller to make a routing decision, increasing the latency of establishing the flow. Subsequent packets are not referred to the SDN controller and are unaffected by this additional latency.

For each round of the latency experiment we send 10 ICMP ping messages to measure latency between two devices or between the device and the WAN (`google.com`). We run 100 rounds of this experiment for each experimental condition, with a 5 second delay between rounds. When testing an SDN app, we cleared all flow rules to ensure that each run reflects the latency of a new flow.

Most latency measurement tools, including `ping`, do not support multicast latency experiments. To evaluate multicast latency, we developed a tool using the Python socket library that measures the time to receive TCP SYN-ACK packets in response to a SYN packet sent to a multicast address. We conduct this test with 2 receiving laptops and one sending laptop. For each experimental condition we run 100 rounds, with a 5 second delay between each round, and clear all flow rules between each run.

For each round of throughput measurements we use the `iperf` tool to conduct a 10-second TCP bandwidth measurement; each round was repeated 20 times for each experimental condition, with a five second delay between experiments. Like the latency experiments, we cleared all installed flow rules between each run.

Our experiment testbed network consists of a standard home router (Linksys WRT 1900 ACSv2) running OpenWRT LEDE 17.01, a Linux desktop serving as the SDN controller, a Macbook Air (Mid 2009) used to generate test traffic, and seven other user devices including smart phones, tablets, laptops, and eBook readers. Because we are evaluating WiFi performance, actual device type will have a negligible effect on latency and throughput. Since Hestia categorizes all IoT devices as either controllers or non-controllers, we performed separate sets of experiments with the Macbook acting as a controller and as a non-controller. Additionally, we configured all of the other 7 devices as the non-controllers in the network so multicast packets are sent to all connected devices by the AP, ensuring uniformity across all experiments. We note that we use the desktop as the SDN controller for convenient deployment; in so doing, we overestimate the network latencies due to the controller. Future deployments can integrate the controller directly into the access point to further reduce latency.

Results: Figure 2 shows the results of our experiments, with the first two subfigures showing latency results. Each bar is shown with an error bar indicating standard error. First, we can observe in Figure 2a that in all communication settings and devices, average latency is largely constant, indicating that Hestia performs on par with both a stock OpenWRT image and a stock SDN app. Naturally, we see that local communications have lower latency than those to the WAN. We also see that multicast packets have drastically higher latency; this is because multicast implementations wait a random amount of time before responding to avoid collisions. The one exception to these trends is in the non-controller multicast latency, which is lower than any other setting because the multicast packet is converted into a series of unicast packets and sent to only a specific set of devices (controllers). This is also an indication that Hestia is providing isolation correctly.

Second, in Figure 2b, we see that the latency for first packets only is significantly higher for all SDN systems, including both the SimpleSwitch app and Hestia, than for the stock access point. This is simply due to the first packet being forwarded to the controller for a flow decision. Hestia performs similarly to SimpleSwitch in all test cases except for non-controller multicast latency, which is faster for the same reason as the previous experiments.

Finally, Figure 2c shows the results of our throughput experiment. Similar to our previous experiments, we find that Hestia does not negatively impact throughput.

6 RELATED WORK

Recent work on securing smart home deployments has focused on traditional mechanisms such as intrusion detection [18], behavioral fingerprinting of IoT devices [5, 11], whitelisting connections at the gateway router [4], or providing new security abstractions by authenticating application data flows on devices [6, 13, 23]. Unfortunately these mechanisms either require a redesign of legacy devices, are not scalable for the constantly evolving smart home ecosystem, or focus solely on selected appified platforms. Prior work has also proposed providing access control to devices based on the context of their environmental situation [19]. However such systems require implementation in the IoT stack of several frameworks from all manufacturers, making it not readily deployable currently.

Closest to Hestia, prior work has provided policy for device communication [4] and monitored device connections using gateway controller devices [11, 15, 23]. Barrera et al. [4] address device-to-cloud communication by whitelisting device connections at the gateway router. However, their fine-grained policy will be difficult to manage as devices gain features allowing customization or extensibility. Since certain smart home products support interfacing with a variety of cloud services, enumerating all possible features in an exhaustive firewall policy will become infeasible. To address the security of device-to-device communication, IoTSec [23] analyzes data flows from IoT devices, and allow interactions based on an application level policy. IoT-Sentinel [15] and IoTurva [11] are designed with a similar architecture in mind. They create ad hoc network overlays for connected devices and use OpenFlow rules to constrain the communication from vulnerable devices. They provide sophisticated ways of identifying a vulnerable device (through an IoT Security Service) and confine its traffic flows using network isolation policies. Their enforcement mechanism, however, only addresses unicast communication from devices. This means that an adversary could still use multicast packets to gain sensitive information from connected devices.

Unlike prior work, Hestia achieves enhanced isolation between devices by mediating service discovery and providing default network policies that are validated for effective deployment. Hestia provides a default access control mechanism for smart home devices, approaching least-privilege, being user-configurable to scale with the changing smart home environment, and simple enough to be readily deployable today.

7 CONCLUSION

The growing number of IoT devices emerging in consumers' home networks continues to raise significant security and privacy concerns. While efforts to improve IoT device security are important, the broad heterogeneity of devices and manufacturers necessitates network-based controls to mitigate the effects of compromised devices. Recently proposed network access controls for IoT devices provide great flexibility, but they require fine-grained policies that are difficult to specify. This paper proposed a practical approach to the problem. We categorized devices into controllers and non-controllers and hypothesized that non-controllers only need to communicate with controllers and cloud servers. We validated this hypothesis using a public dataset of network traces of over 40 smart home devices. We then proposed Hestia, which enforces this intuitive policy. Hestia only requires users to specify which devices are controllers and introduces negligible performance overhead. As such, Hestia provides an effective and practical way to provide least-privilege access control in smart home networks.

8 ACKNOWLEDGMENT

We would like to thank our anonymous reviewers for their valuable comments, and Merve Sahin for shepherding our paper. We also thank Terrence O'Connor, Benjamin Andow, Issac Polinsky, and the Wolfpack Security and Privacy Research (WSPR) lab as a whole for their helpful feedback.

REFERENCES

[1] Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. 2018. Web-based attacks to discover and control local IoT devices. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*. ACM, 29–35.

[2] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security Evaluation of Home-Based IoT Deployments. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 0.

[3] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *USENIX Security Symposium*.

[4] David Barrera, Ian Molloy, and Heqing Huang. 2018. Standardizing IoT Network Security Policy Enforcement. In *Workshop on Decentralized IoT Security and Standards (DISS)*, Vol. 2018. 6.

[5] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. 2018. Behavioral Fingerprinting of IoT Devices. In *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security (ASHES '18)*. ACM, New York, NY, USA, 41–50. https://doi.org/10.1145/3266444.3266452

[6] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A. Gunter, Xiaoyong Zhou, and Michael Grace. 2017. HanGuard: SDN-driven Protection of Smart Home WiFi Devices from Malicious Mobile Apps. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '17)*. ACM, New York, NY, USA, 122–133.

[7] Nitesh Dhanjani. 2015. *Abusing the internet of things: Blackouts, freakouts, and stakeouts*. O'Reilly Media, Inc.

[8] Ben Dickson. 2019. More Smart home devices vulnerable, McAfee researchers find. Retrieved March 15, 2019 from https://www.dailydot.com/debug/smart-home-mcafee-attacks

[9] F-Secure. 2018. What are the current projection features for f-secure sense? Retrieved January 5, 2019 from https://community.f-secure.com/t5/F-Secure-SENSE/What-are-the-current-security/ta-p/82972

[10] Andy Greenberg. 2017. Hack Brief: 'Devil's Ivy' vulnerability could afflict millions of IoT devices. Retrieved March 15, 2019 from https://www.wired.com/story/devils-ivy-iot-vulnerability/

[11] Ibbad Hafeez, Aaron Yi Ding, and Sasu Tarkoma. 2017. IOTURVA: Securing Device-to-Device (D2D) Communication in IoT Networks. In *Proceedings of the 12th Workshop on Challenged Networks*. ACM, 1–6.

[12] Seppo Hätönen, Petri Savolainen, Ashwin Rao, Hannu Flinck, and Sasu Tarkoma. 2016. Off-the-Shelf Software-defined Wi-Fi Networks. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 609–610. https://doi.org/10.1145/2934872.2959071

[13] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. 2017. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms.. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*.

[14] Eliot Lear, Ralph Droms, and Dan Romascanu. 2018. *Manufacturer Usage Description Specification*. Internet-Draft. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-ietf-opsawg-mud-25.txt

[15] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. 2017. IoT Sentinel: Automated device-type identification for security enforcement in IoT. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2177–2184.

[16] Lily Hay Newman. 2018. An Elaborate Hack Shows How Much Damage IoT Bugs Can Do. Retrieved March 15, 2019 from https://www.wired.com/story/elaborate-hack-shows-damage-iot-bugs-can-do/

[17] phikshun. 2018. Belkin Netcam HD UPnP Command Injection - Github. Retrieved January 5, 2019 from https://gist.github.com/phikshun/9984624

[18] Shahid Raza, Linus Wallgren, and Thiemo Voigt. 2013. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad hoc networks* 11, 8 (2013), 2661–2674.

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

[20] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. 2016. Smartphones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 195–200.

[21] Statista. 2018. Smart Home Market penetration in US. Retrieved January 2, 2019 from https://www.statista.com/outlook/279/109/smart-home/united-states

[22] Wang Wei. 2018. Casino Gets Hacked Through Its Internet-Connected Fish Tank Thermometer. Retrieved March 16, 2019 from https://thehackernews.com/2018/04/iot-hacking-thermometer.html

[23] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. 2015. Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets-XIV)*. ACM, New York, NY, USA, Article 5, 7 pages. https://doi.org/10.1145/2834050.2834095