Can Allowlists Capture the Variability of Home IoT Device Network Behavior?

Abstract

Home IoT devices are replete with unpatched security vulnerabilities. Because these devices can be difficult for users to secure and patch, network-based defenses are promising. In contrast with prior work, which primarily studied blocklists, we instead study the creation of host-based *allowlists* for improving IoT network security. Allowlists are appealing because they require minimal device configuration and leave a small attack surface, yet are challenging to implement for general-purpose computing devices because of the heterogeneity in devices and the destinations they contact. However, intuition suggests that home IoT devices actions' might be limited enough for allowlists to be practical. In this paper, we evaluate the feasibility of allowlists in this setting in several ways. To understand the variability of home IoT devices and the impact on allowlists, we first contrast the network behaviors of thousands of IoT devices collected in the IoT Inspector dataset, finding a surprising amount of variability in the destinations contacted. A key challenge is to derive allowlist rules that limit the attack surface while capturing enough of this variability to "transfer" across households and devices. Thus, we then explore the potential design space for allowlists, quantifying the impact of various ways of representing hosts, requiring a host to appear in multiple devices' traffic, and various sample sizes. We find that a partially abstracted representation of hosts may best balance security and transferability. Finally, to understand whether allowlists can be deployed without hindering device functionality, we test various allowlists on popular devices in our lab. In sum, we find that home IoT device allowlists may be practical in many circumstances if care is paid to these design considerations. The allowlists generated appear relatively stable over time and, for many products, do not seem to impair functionality.

1 Introduction

The majority of homes in many countries now contain Internet of Things (*IoT*) devices, such as smart lightbulbs, cameras, voice assistants, or thermostats [33]. Unfortunately, home IoT devices have suffered from numerous security shortcomings [7]. Attackers have exploited vulnerabilities in home IoT devices' software, protocols, and default settings to take control of devices [10] for purposes including creating botnets like Mirai [8] and Hajime [22]. Contributing to these security issues is the wide variety of (often inexperienced) manufacturers creating home IoT devices, the difficulties of deploying software patches to devices that may not have screens or traditional user interfaces, and the lack of standardization [36].

Rather than relying on potentially unresponsive vendors to patch devices, an appealing solution is for a household to monitor the network traffic of all of its home IoT devices, applying broad security policies designed to disallow problematic network behaviors, such as potential distributed denial of service (*DDoS*) attacks or the exfiltration of data about the home to potentially illegitimate endpoints.

One important defense is to use firewall-style rules to specify whether a particular home IoT device should be permitted to contact a particular endpoint. Recent papers [16, 24, 29] have employed *blocklists*, which specify (problematic) hosts that cannot be contacted and permit traffic to all other hosts. From the security perspective, an even more attractive approach would be to employ *allowlists*, which instead enumerate the hosts that can be contacted and block traffic to all other hosts. While allowlist-based approaches have a much smaller attack surface, they are infrequently used in practice because enumerating the destinations that general-purpose computing devices should be able to contact is typically intractable. On the other hand, many home IoT devices typically have a highly limited set of actions and behaviors. Intuition thus suggests that allowlists may be practical for securing home IoT devices at the network level.

In this paper, we explore the design space of creating allowlists for home IoT devices, answering a number of open questions in the process. While some prior papers have proposed allowlists for IoT devices in one form or another [17, 20, 21], they have evaluated this idea only on a single device in a lab setting. If attempting to create a gen-

eralized allowlist that transfers to other users, studying only a single device in a controlled environment is likely to miss important sources of variability across devices, across regions, across network architectures, and across time. Alternatively, requiring each user to create a personalized allowlist can require an onerous time investment for users who would need to exercise all of their devices' functionalities, and creating an allowlist based on a single device that has already been compromised would nullify any security guarantees. To solve this problem, many of our experiments rely on data shared with us by the IoT Inspector project [23], which collected network traffic data from 5,439 unique verified home IoT devices from real-world household deployments. This approach enables us to analyze how the network behaviors of a single *product* (e.g., the Philips Hue lightbulb) varies across *devices* (e.g., a single household's physical instance of a Philips Hue) in different homes across the world.

First, in Section 4, we used the IoT Inspector dataset to investigate our initial assumption of whether home IoT devices do actually have relatively homogeneous network behaviors, especially the degree to which network behaviors vary across devices of the same product (e.g., variation across different households' Philips Hue lightbulbs). We focused our analysis on the hostnames (i.e., fully qualified domain names) with which each device communicates. We expected to see variations based on network-specific configurations (e.g., DNS and NTP settings), which we indeed observed. To our surprise, we also observed substantial variation in the hostnames contacted by different devices of the same product far more broadly than expected, beyond configuration-specific facets. On average, to observe 95% of the hostnames contacted by all devices in our dataset of a given product required a sample of over 60% of the devices of that product. Nonetheless, many hostnames in this "long tail" were related to each other. Some hostnames contained what appeared to be randomized substrings (e.g., `oculus2975-us1.dropcam.com` vs. `oculus1802-us1.dropcam.com`) presumably to support load balancing, while others contained what appeared to be regional identifiers (e.g., `avs-alexa-6-na.amazon.com` vs. `avs-alexa-7-eu.amazon.com`). As a result, an allowlist of hostnames created by analyzing the network traffic of one specific device is surprisingly unlikely to generalize to other devices of that same product, necessitating more nuanced generation of allowlists.

In Section 5, we thus evaluate the design space for allowlists. First, we compare different representations of hosts. Compared to an allowlist of hostnames, an allowlist of domains (i.e., second-level domains, or 2LDs) is far more likely to capture this legitimate variability and thus transfer across devices of the same product. Unfortunately, many IoT devices rely on cloud services (e.g., `amazonaws.com`), meaning that domain-based allowlists open a large attack surface. As a compromise, we propose and evaluate a pattern-based representation of hostnames, which appears to enable transferring allowlists across devices while limiting the attack surface. Similarly, to make allowlists more robust in the presence of a small number of mislabeled or compromised devices, we explore requiring that a host be contacted by n different devices (termed the *threshold*) for it to be added to the allowlist. We also investigate the effect of the number of different devices of a product necessary to create a robust allowlist, finding it to be on the order of one or two dozen devices, depending on the host representation.

The aforementioned experiments were retrospective simulations on the IoT Inspector dataset, raising the question of whether allowlists created from this dataset would actually work in practice. Thus, in Section 6, we report on an in-lab experiment, where we tested the functionality of nine popular home IoT products using various types of allowlists generated from the IoT Inspector dataset. When deployed behind a network firewall with our allowlists, some actions (e.g., streaming music or video, downloading apps) occasionally failed, yet most other actions we tested worked normally under the allowlists tested. While we had hypothesized rare actions (e.g., reinitializing the device) might fail to function because they likely had not been captured in the IoT Inspector dataset, we observed that these rare actions typically functioned normally. In other words, snapshots of two-year-old network traffic data from other individuals' devices can be sufficient for generating allowlists for home IoT devices. We conclude by discussing lessons learned about generating allowlist-based security mechanisms for home IoT devices.

2 Problem Setting and Key Terminology

In our setting, we imagine that there is a user who has several home IoT devices connected to their home network, over which they have total control (e.g., for deploying an allowlist). In our threat model, a remote attacker aims to compromise these IoT devices, forcing them either to contact a server under the attacker's control (e.g., for data exfiltration or botnet command and control) or to direct traffic toward some victim (e.g., as part of a distributed denial of service, or *DDoS*, attack). The attacker always has the ability to launch attacks from a new server. However, they do not have the ability to compromise the IoT device vendor's backend infrastructure (e.g., `belkin.com`), nor poison the victim's DNS. Local attacks (e.g., on WiFi or ZigBee) are out of scope.

The user (defender in our threat model) aims to create an **allowlist**, or specification of the endpoints with which a given home IoT device is permitted to communicate. For instance, an allowlist might specify that a given home IoT device can only communicate with `belkin.com`; all other traffic, both inbound and outbound, is blocked. In a given home, different products will likely have different allowlists, but multiple devices in the home of the same product (e.g., all six of the user's Wyze Cameras) will share the same allowlist. We assume the contents of the allowlists are known to both

the defender and the attacker.

In this paper, we create allowlists based on a large-scale dataset of IoT network traffic from many devices, the IoT Inspector dataset (see Section 3). We test the allowlists generated on held-out data from the same dataset in Section 5, and on real devices in our lab in Section 6.

In describing the process of allowlist creation, we use the following terms. **Vendor** refers to a company that makes home IoT products (e.g., Amazon). **Product** refers to the collection of devices sold under a specific name (e.g., Amazon Ring), potentially encompassing multiple releases and versions. **Device** refers to a single physical object that is an instance of a home IoT product. For example, a single Amazon Ring in someone's home is a single device. **Type** refers to the category of multiple home IoT products with similar purposes. For example, both the Amazon Ring product and Wyze Camera product are of the type "home IoT camera."

In Section 5, we analyze three main design dimensions for allowlists. First, we compare different **host representations**, or abstractions for characterizing endpoints. For instance, these representations include specifying allowed endpoints through lists of hostnames, domains, IP addresses, and more (see Section 5.3). Second, we compare different restrictiveness **thresholds**; for a threshold of n, we only add a particular host (per the host representation being tested at the time) to an allowlist if it appears in the traffic of at least n devices in the sample of devices being used to construct the allowlist. Using a threshold aims to make allowlists more robust in the face of noise, mislabeled data, and potentially a small number ($< n$) of compromised devices being included in the sample. Third, we compare different **sample sizes**, or sets of unique devices (i.e., from different households) of a single product being used to generate an allowlist.

3 Dataset

The dataset we use throughout this paper to analyze the variability of devices' network traffic (Section 4), generate allowlists (Sections 5–6), and retrospectively analyze the impact of different allowlist design decisions through simulations (Section 5) is the IoT Inspector dataset [23], which the authors of that work shared with us. Specifically, we analyze the relevant subset of IoT Inspector containing only devices with validated product names and network traffic recorded. This subset consists of 5,439 unique devices, representing 424 unique products from 80 different vendors among 43 device types. These 5,439 unique devices came from 1,468 households in total. Data collection occurred between April 8 and July 24, 2019, documenting 38,164,870 network flows in total. We define a network flow as the combination of the device ID, local port, and remote IP address and port [23].

For clarity, in the body of the paper we report on the 20 products that appeared most frequently in the dataset; there were 3,456 devices among these 20 products in the dataset. The appendix presents analogous graphs with the 52 products for which there was enough data to perform those analyses.

The dataset contains the following information, some of which involved post-processing by the IoT Inspector authors. Each flow in the dataset contains the **product and vendor** of the home IoT device whose traffic was being monitored; this information was entered manually by participants on the IoT Inspector user interface. Each flow also contains the **remote IP address and remote port** contacted by the home IoT device. The original authors attempted to compute additional information about the remote destination via a best-effort process detailed in the original paper that included examining DNS traffic. They attempted to associate the remote IP address with a **hostname** (fully-qualified domain name) and **domain** (top-level and second-level domain, such as `google.com`). Each flow is also associated with the household's **timezone**, as opposed to a more precise geo-location, to respect participant privacy.

As with any dataset collected in a real-world deployment, data cleaning was necessary. While the original authors performed initial data cleaning [23], we further cleaned the data in two ways to better support our analyses. First, because product names were manually entered by participants, they may not be reliable. We regard a device's product name as likely mislabeled if the device never contacts any domain that other devices with the same product name commonly contact. We considered a domain to be common for that product if at least 20% of devices for the product have contacted it, an arbitrary value we chose to reduce the likelihood of mislabeled devices. Later in the paper, we use thresholds to further account for potentially mislabeled devices. We made an exception for devices that only contact DNS or multicast DNS (mDNS) resolvers because these may be devices that are simply not used by the participants during the data collection period. We found 16 potentially mislabeled devices across 10 products and excluded them from further analysis. We also manually examined other outliers. We found one device, labeled as a Belkin Wemo switch, that contacted 26,594 unique domains. This behavior is very unusual based on how other Belkin Wemo switches, or even any other home IoT devices, behave. We suspected this device might be either compromised or mislabeled, so we removed this one device from the dataset.

Second, we also aimed to identify hostnames and domains for which the original DNS lookup may not be in the IoT Inspector trace. Although most traffic flows on the Internet are preceded by a corresponding DNS lookup, IoT devices are already installed and running by the time a user may run IoT Inspector, the dataset may not see all DNS traffic associated with a traffic flow, such as the initial DNS lookup from the device for domain name associated with a traffic flow. Although the original authors of the dataset guessed missing hostnames (those missing associated DNS traffic) using passive traffic monitoring and reverse DNS lookups, we determined these methods to be potentially *unreliable*.

For example, a reverse DNS lookup from an IP address often resolves to particular infrastructure, such as a server, rather than a general hostname. About 62.1% of the collected flows contained these potentially unreliable hostnames. To further validate the data, we first compared the IP addresses of any two packets—one known to have the correct hostname based on associated DNS queries in the trace the IP address without an associated lookup—that shared the same hostname. If the second packet had the same IP address as the first one, then we concluded that it had the correct hostname. To mitigate the effects of virtual hosting, whereby different domains can map to the same IP address, we first performed this process at the product level, then repeated it at vendor level. This process reduced the fraction of flows with unreliable hostnames to 34.4%. We repeated this approach using ASNs instead of IP addresses. This step left us with 30.4% of flows that has an unreliable hostname. These unreliable hostnames are still kept for further analysis to avoid missing data, but we also retain an annotation about their unreliability.

4 Variability of Device Behavior

Ideally, a user would have complete knowledge about a product's behavior when creating an allowlist, but unfortunately vendors typically do not provide that information. With the help of the IoT Inspector dataset, we instead use observed network traffic from one or more devices of a given product to create potential allowlists. However, this approach of transferring observations of one device's network behaviors to another device of the same product (ostensibly owned by a different person and on a different network) will only work if the allowlist can capture the differences in network traffic across devices of a given product. Thus, in this section, we use the IoT Inspector dataset to analyze how devices of the same product vary in their network behavior. We find substantial variation for some home IoT devices. As detailed in this section, variability appears to arise for many reasons, including load balancing and caching, under-represented regional services, user-defined DNS resolvers, and hosts that are not associated with a known hostname or domain.

4.1 Coverage of Network Behaviors

As a first step in understanding variability, we took the union of the hosts seen across all devices for each of the 20 most common products in the IoT Inspector dataset and analyzed what fraction of those hosts were observed in samples of devices for that product. We use the term **coverage** to refer to the percentage of flows (remote host and remote port) contacted by any device (of that product) that were contacted by a given sample of those devices. If devices behaved relatively similarly to each other, coverage would be close to 100% for even a small sample of those devices.

For many products, however, this was not the case. Figure 1 shows the coverage of different sample sizes for three different granularities of representing hosts: domains (e.g., `dropcam.com`), hostnames (e.g., `oculus1802-us1.dropcam.com`) and IPv4 address (e.g., `35.186.28.155`). For each sample size, this figure shows the average coverage over 100 random samples. For most of the 20 products, the coverage of domains approaches 100% even for small samples. However, for the majority of these products, the coverage for hostnames and IP addresses was substantially lower than for domains at a given sample size. This result suggests that, for many products, a given device may be contacting different hostnames and IP addresses, even when it is contacting the same domain, creating challenges for more granular allowlists.

While, due to load balancing, we expected to observe lower coverage for IP addresses, we were more surprised that hostnames followed a similar pattern for some products (e.g., Google Nest and Amazon Ring). Averaged across all 20 products, coverage of 95% of flows was only achieved with a sample of 58% of the devices of that product in the dataset when using hostnames, compared to 67% when using IP addresses. The percentages increase to 81% (hostname) and 88% (IP address) for 99% coverage. That coverage is far from 100% for small samples supports our assumption that prior work that considered allowlists based on only a single device [17, 20, 21] is likely to miss important variabilities and thus fail to create allowlists that transfer across devices.

4.2 Sources of Variability

Creating allowlists that transfer across devices requires further understanding potential root causes of this variability.

Load Balancing and Caching: One of the main sources of variability appears to be the use of different, but related, hostnames to support load balancing and caching. For example, Apple Push Notification (APN) service sets up multiple servers for better availability, using hostnames like `27-courier.push.apple.com`, where the number (27) varied across devices in our dataset. When a large number of similar hostnames exhibit small variations, an allowlist created based on precise hostnames is unlikely to generalize.

The good news, however, is that most of these hostnames follow a specific pattern. In the simple example above, the number is the only part of the the hostname that we observed changing, which suggests the possibility of creating a more generalizable, abstract representation of the hostname (see Section 5.3). Some other hostnames varied in more than just numbers. For example, we observed many Spotify hostnames that take forms like `guc3-accesspoint-a-f002.ap.spotify.com`, where `f002` is the substring that varied.

Regionalization: Geographical bias in the IoT Inspector

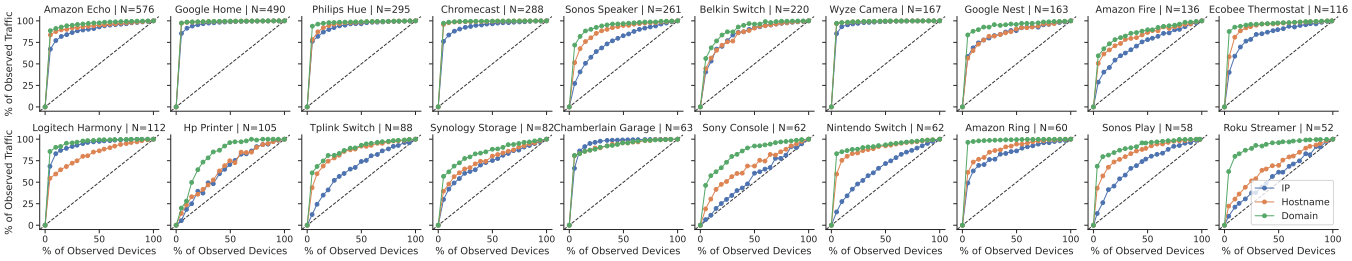

Figure 1: Flow coverage (averaged across 100 runs) for three ways of characterizing endpoints (domain, hostname, IP address) for the 20 most popular products. Coverage well below 100% for small samples indicates high variability across devices.

dataset, which was collected by US-based researchers using an tool documented in English, is an additional consideration. In the subset of IoT Inspector we analyzed, 70.1% of devices appear to be in North or South America based on timezone.

However, we observed that devices in different geographic regions sometimes contact different endpoints. For example, Amazon Echos in North America ("NA") tended to contact hosts like `avs-alexa-6-na.amazon.com`, whereas those in Europe ("EU") tended to contact hosts like `avs-alexa-7-eu.amazon.com`. Some variability was far more subtle than simply replacing region codes in hostnames. For example, Belkin Wemo switches in North and South America typically contacted `navy.mil` for network time protocol (NTP) services, while those outside North and South America usually contacted `tu-berlin.de` instead.

The destinations that users visit on home IoT devices that stream media also varies across regions, reflecting local interests but making allowlists less likely to generalize across regions. For example, we observed endpoints like `tf1.fr` (a French television channel, contacted by a Google Chromecast), `bell.ca` (an ISP in Canada, contacted by an Amazon Echo and a Chromecast), and `met.no` (a weather service from Norway, contacted by a Synology network storage device to display weather information on the user interface). These observations highlight the importance of regionalization when generating allowlists.

DNS: Because users (or their ISPs) typically assign a local DNS resolver, DNS traffic was an additional (expected) source of variability. We observed a long tail of DNS traffic, including 102 unique DNS resolvers contacted by only one device.

Other Causes: Other variability may be artifacts of IoT Inspector's data collection process. For example, 26.7% of endpoints are IP addresses without any associated hostname or domains. This may be caused by IoT Inspector missing the relevant DNS traffic, or because these hosts are actually contacted using hardcoded IP addresses. In fact, we observed the latter in our own experimentation when the Wyze camera in our lab contacted some hosts without sending out any DNS queries. Prior work [15] also found, for instance, that Belkin Wemo plugs transmitted IP addresses directly in payloads at

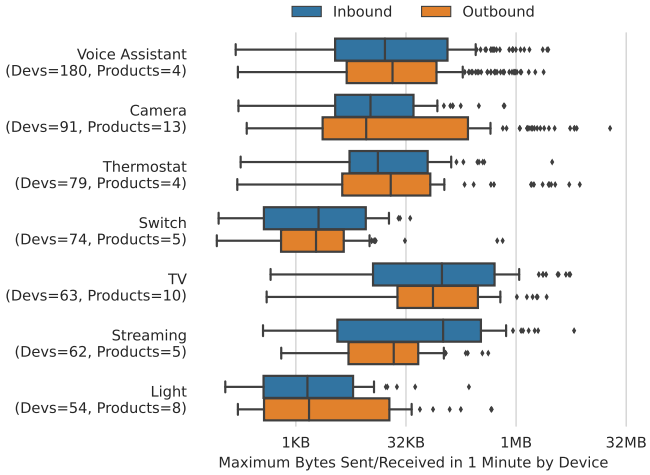

Figure 2: Distributions of the maximum bytes that a given device sends (receives) within 1 minute, grouping by type.

the time of their data collection in 2020, though our experiments found that they now transmit hostnames in payloads.

Some smart home products, such as the Amazon Echo, have multiple generations. We grouped these generations into a single product. It is possible that different generations of some products may behave differently. Even for the exact same product generation, behavior may change depending on the software or firmware version. As we detail in Section 6, a recent update to Belkin Wemo smart plugs caused them to contact a completely different set of endpoints, presumably due to updates in Belkin's backend infrastructure. Furthermore, because product names were provided manually by participants and some gave only a very general name (e.g.,"Sonos speaker"), it is also unclear to us if all Sonos speakers share the same infrastructure.

4.3 Throughput by Device Type

Although the rest of this paper focuses on the endpoints home IoT devices contact, we also examined the variability in the amount of traffic they send and receive. In cases like the Mirai botnet [8], IoT devices were repurposed for DDoS. Thus, characterizing and perhaps throttling the amount of traffic

they could send is an additional network-level defense.

We expected that, while products of the same type may not share similarities in the remote hosts they contact, they might require similar throughput, which could imply throttling limits. Figure 2 thus shows, for a larger set of products, the maximum amount of traffic any given device in our dataset in any given one-minute period. The figure only includes flows with an MTU $\leq$ 1500.[1] Figure 2 shows that TVs in our dataset received up to 5.5 MB of data per minute, while switches received up to 36.5 KB per minute at most. As expected, switches and lights receive and send much less data than other types. However, we observed some surprising outliers. For example, a TP-Link switch and a Belkin Wemo switch respectively sent 671.12 KB and 565.55 KB in one minute to `n-devs.tplinkcloud.com` and `nat.xbcs.net`, respectively. We contemplate that these outliers may be caused by some rare user operations or by vendor's own data collection. A higher throttling threshold can be placed on the device if one would like to ensure full functionality. On the other hand, TVs and streaming devices generally receive more traffic than others. One interesting observation is that cameras are the only ones that send much more data than receive. The amount of data cameras sent in one minute also varies a lot, from tens of bytes to 19.28 MB, making them harder to throttle.

5 Generalizability of Allowlists

To investigate how the design space of allowlists impacts the network traffic permitted, we must first devise a method for producing allowlists that can reflect various design factors. We develop a proof-of-concept algorithm that generates allowlists by varying the host representation (granularity and abstraction of endpoints), thresholds, and training and testing samples (with constraints such as sample size, geo-location, and attempts to transfer across products).

5.1 Methods

We introduce four variables for allowlist generation. **Host representation** describes the level of abstraction for describing hosts. In addition to our focus on domains, hostnames, and IP addresses in the body of the paper, the appendix also compares (less successful) formats, such as subnet or BGP prefix. As previously defined, **threshold** is an integer that specifies the minimum number of devices on which we need to observe an endpoint to add it to the allowlist. **Training data** references the dataset used to generate the allowlist. It can be chosen based on criteria like the product or timezone. **Testing data** references the dataset used to evaluate the generated allowlist. It can be chosen by similar criteria. In all cases, a single device cannot be in both the training data and testing data for a given experiment.

As further discussed in Section 5.3, one hostname representation we consider, the **pattern** representation, provides a partial abstraction of the hostname. To generate these patterns, which are effectively regular expressions, we use DBSCAN to group similar hostnames for each product. We cluster each hostname based on its lowest level subdomain (i.e., leftmost part of the FQDN) and use the remaining part of the FQDN for grouping so that different domains or subdomains are not clustered together. We condense any consecutive numbers to "[0-9]+" because digits rarely carry any meaning in this context. Once a cluster has been identified, we generate an ordered list of the longest non-overlapping common substrings for all hostnames in the cluster. If we can further group the remaining parts, we accept these variations into the allowlist rather than using a wildcard to minimize the attack surface. If the remaining parts appear random, a wildcard will be placed in their place. For example, hostnames used by Apple Notification Services that we mentioned in Section 5 become *"[0-9]+-courier.push.apple.com"*, while access-point hostnames of Spotify become *"guc3-accesspoint-a-.*.ap.spotify.com"*.

5.2 Key Metric

We evaluate potential design choices for allowlists through a retrospective simulation based on the IoT Inspector dataset. For a given product, we calculate a value we term the **median fraction of allowed flows (MFAF)**. Intuitively, the MFAF captures the fraction of traffic previously observed without the allowlist that would still have been permitted with the allowlist active. Thus, a high MFAF is desirable since it reflects the device's observed traffic proceeding as normal. We take the median fraction of allowed flows across devices instead of the mean because the distribution can be very skewed.

More formally, MFAF is defined as:

$$MFAF(A_D, D') = median(\{\frac{|F^d_{A_D}|}{|F^d|}, \forall d \in D'\}) \quad (1)$$

where D is the group of devices used for generating allowlist A_D, D' is the group of devices that the allowlist is applied to, d is a device in D', F^d is the all the flows transmitted on d, and $F^d_{A_D}$ is the fraction of d's flows allowed when the allowlist A_D is active. We require $D \cap D' = \emptyset$ for a fair evaluation. A flow is identified by a combination of the device ID, the local port, and the IP address and port of the remote endpoint.

5.3 Host Representation

We tested seven possible host representations. In this section, we contrast representing endpoints by their domain, pattern (see Section 5.1), and exact hostname. We present the other four alternatives, which were less successul, in the appendix.

As shown in Figure 3, all 20 popular devices had high MFAF (greater than 0.9) when using domain representations.

[1] We impose a limit on the MTU as some packet sizes collected by IoT Inspector significantly exceed the MTU, up to millions of bytes per packet. We observe these apparently erroneous MTU values on 2.4% of devices.

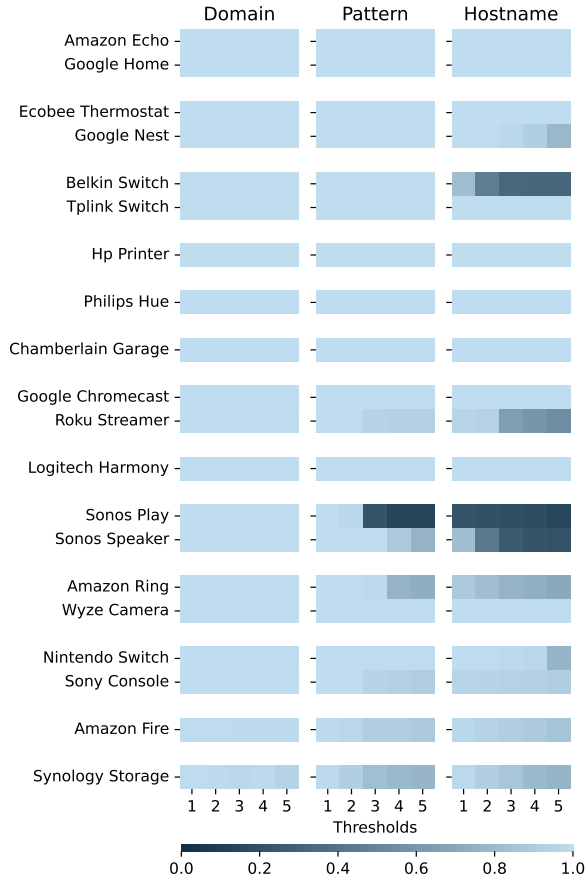

Figure 3: MFAF of allowlists generated from, and applied to, the same product in a train-test split. The products (rows) are grouped by type. The color indicates the proportion of flows permitted using a threshold of between 1 and 5 (sub-columns).

While this means that the (presumably legitimate) traffic observed in our test set would be allowed, domain-based allowlists also have the largest attack surface, as we revisit later in this section.

In contrast, both hostname and pattern representations, particularly the latter, seemed to better balance generalizability and security. With the threshold set to 1, hostname representations achieved an MFAF greater than 0.9 (*mean* = 0.97) for 18 of 20 products. The two exceptions were the Belkin Switch and Sonos Play. Using pattern representations, however, all 20 products again achieved an MFAF greater than 0.9 while maintaining a relatively small attack surface.

Pattern representations also had clear advantages over hostname representations at higher thresholds for a few products. A higher threshold means a stricter allowlist, which makes the allowlist more robust to noisy or poisoned data. When threshold is set to 5, only 10 products have an MFAF of at least 0.9 when using hostname representations, but 15 products have an MFAF of at least 0.9 when using pattern representations. Emphasizing these trends, Figure 4 shows the six products whose MFAF increases more than 0.1 at least once by switching from hostnames to patterns. Appendix C shows that similar trends hold when analyzing a larger dataset of 52 products.

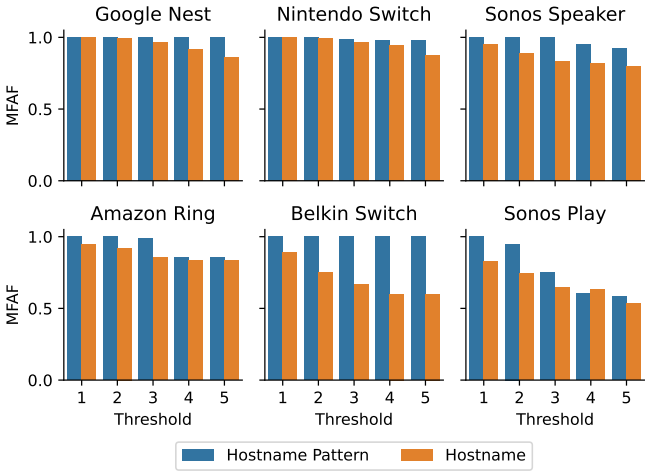

Figure 4: MFAF changes after switching from hostname-based allowlists to pattern-based allowlists.

Examples (Hostnames w/ Patterns and Domains)	Host #
arlostatic-z1.s3.amazonaws.com	1
• *arlostatic-z[0-9]+\.s3\.amazonaws\.com*	3
• *amazonaws.com*	3859
ring-untranscoded-videos.s3.amazonaws.com	1
• *ring-(|un)transcoded-videos(-lq|)\.s3\.amazonaws\.com*	2
• *amazonaws.com*	3859
a191avoddashs3ww-a.akamaihd.net	1
• *a[0-9]+avoddashs[0-9]+(ww|us)-a\.akamaihd\.net*	50
• *akamaihd.net*	96

Table 1: Examples about how host representations can impact the attack surface. The rightmost column shows the number of hostnames in the IoT Inspector dataset that will be allowed if one puts the destination on the allowlist.

Security Analysis: While domains are more generalizable than hostnames or patterns, they can easily fall short in security, particularly due to home IoT devices' reliance on cloud providers. Table 1 shows three example hostnames, contrasting with pattern and domain representations. For example, `arlostatic-z1.s3.amazonaws.com` is one of the hostnames used by Netgear's Arlo cameras. We observed the last digit of the lowest level subdomain changed to other numbers in the dataset. Whereas the pattern abstracted from this single hostname matches three different hostnames in the dataset, extending to the domain `amazonaws.com` matches 3,859. A similar analysis can be extended to the space of possible, rather than observed, hostnames matching a pattern or domain, particularly concerning whether an attacker is able to register an illegitimate endpoint, or attack some victim's endpoint, matching the pattern (less likely) or domain (far more likely). In this example, relying on domains, rather than

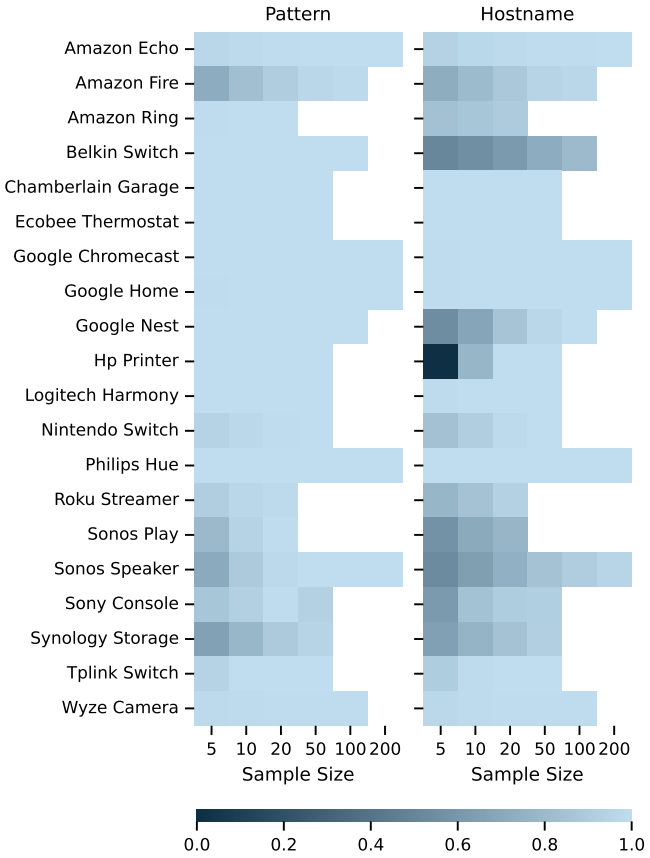

Figure 5: The MFAF of allowlists increases with more devices involved in training (*threshold* = 1). White spaces mean the given sample size is not applicable (not enough training data).

patterns, brings all of `amazonaws.com` into the threat model. The attacker can easily host malicious content on AWS or direct a DDoS attack toward victims using AWS. Using a pattern often provides a far narrower attack surface than using a domain, while still supporting generalizability.

That said, hostname patterns can have complex security implications. For *arlostatic-z[0-9]+\.s3\.amazonaws\.com*, an attacker can exploit the fact there is not limitation placed on the numbers and register a hostname like *arlostatic-z111.s3.amazonaws.com* to pass the regular expression.

5.4 Sample Size Required

Using a larger sample of devices for a given product when generating an allowlist makes it increasingly likely that the allowlist will encompass the long tail of endpoints with home a product communicates. Thus, we calculated the MFAF for various sample sizes. Specifically, we altered the size of the training sample with a step size of five, observing how the MFAF changed. For stability, we performed five-fold cross validation on each product, and we also repeated this process five times, taking the mean.

The results are shown in Figure 5. In general, hostname representations necessitate a larger sample than pattern representations. For hostname-based allowlists, only 15 out of 20 products ever achieved an MFAF of at least 0.95, and it took 28 devices on average to do so. In contrast, pattern-based allowlists enabled all products to achieve an MFAF of at least 0.95, requiring only 12 devices on average. The standard deviation of these numbers, however, was high, particularly for hostname representations. For example, 5 Ecobee Thermostats were sufficient to form a hostname-based allowlist, but 185 Sonos Speakers were needed for the same MFAF.

Pattern representations were also important when the sample size was necessarily small due to the unavailability of data. For example, the dataset contained only a handful of HP printers, and HP uses load balancing for XMPP communication (e.g., with `xmpp001.hpeprint.com` and `xmpp002.hpeprint.com`). With at least two related hostnames observed, pattern representations can thus potentially compensate for a smaller sample.

5.5 Setting the Threshold

The threshold, specifying on how many different devices an endpoint must be observed for it to be added to the allowlist, represents a direct trade-off between security and generalizability. Increasing the threshold typically admits fewer endpoints, shrinking the attack surface, and increases robustness to a few devices being mislabeled or compromised.

Interestingly, we observed that increases in the threshold were not proportional to the decrease in the amount of traffic allowed. For some products, it was possible to have a high threshold without losing much generalizability. To understand the relationship between thresholds and the MFAF, we sampled 50 devices from each product, calculating the MFAF under thresholds ranging from 1 to 39. The process was repeated 5 times, taking the mean.

As shown in Figure 6, some products effectively had plateaus in which increasing the threshold minimally impacted the MFAF. In one of the more extreme examples, the MFAF for the Google Home was similar between thresholds of 1 and 25. One possible explanation is that, upon increasing the threshold, only the most fundamental endpoints contacted by nearly all devices of that product are kept, and most flows are to those endpoints. For Chamberlain Garage, most endpoints in the long tail were various DNS resolvers contacted by few devices. Only two endpoints were contacted by the majority of those devices: `connect1.myqdevice.com` and `connect.myqdevice.com`. In contrast, products like the Amazon Fire and Roku Streamer that rely on streaming user-specied content rarely exhibit this plateau.

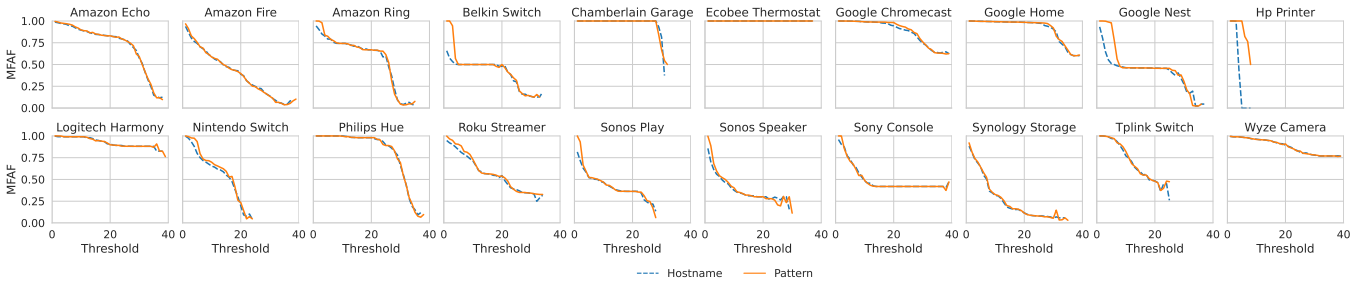

Figure 6: Although the MFAF decreases with increasing thresholds, many products experience plateaus where the thresholds increase a lot, but the corresponding MFAF does not drop much. Lines end when the allowlist is blank above that threshold.

5.6 Accounting For Regionalization

As previously mentioned, IoT Inspector primarily consists of devices from North America and South America. To better understand how geography and regionalization impact allowlists' MFAF, we created a series of regression models that gauge how region impacts MFAF. Because the IoT Inspector data set doesn't contain location information about the devices, we use timezones as a proxy, creating three regions: **Region A** (UTC-02:30 - UTC-10:00), **Region B** (UTC+01:00 - UTC+04:00), and **Region C** (UTC+5:30 - UTC+12:00). We selected the six products in the dataset that each have at least 10 devices in each region: Amazon Echo, Belkin Switch, Google Chromecast, Google Home, Philips Hue, and Sonos Speaker. We created linear regression models in which the MFAF was the dependent variable and the regions of the training and testing samples, the product, and the (log of the) sample size were the independent variables.

At a high level, applying an allowlist to testing data from a region different from the region of its training data negatively impacts the MFAF. Furthermore, regions differ in the MFAF of the allowlists generated even when keeping the sample size constant. Table 12 in the appendix contains the regression results, and all coefficients of the train region, test region, and their interactions are significant. For example, applying allowlists generated from devices in Region A to devices in Region C causes a drop of around 0.1 in MFAF. Therefore, if the data is geographically diverse, care should be paid to matching the training data and test data as the network behaviors of home IoT devices are partially regionalized.

5.7 Transfering Allowlists By Vendor

To this point, we have created allowlists for a product based on traces of the network behaviors of devices of that same product. While this is the most typical setting, for newly introduced or rare devices, there may not be appropriate data from which to create an allowlist. This situation raises the question of whether it is possible to transfer an allowlist generated for one product to a *different product made by the same vendor*. For example, could an allowlist generated for Google Home work successfully if applied to Google Nest devices?

At a high level, we found that domain-based allowlists can sometimes be transferred to other products from the same vendor. Exceptions we observed include IoT cameras and cases where the vendor makes a wide range of products. Furthermore, pattern-based and hostname-based transferred somewhat less well across products, though was often still possible.

Figure 7 was created by using all other products from the same vendor as training data and using the product listed as the testing data for computing MFAF. Comparing Figure 3 and Figure 7 shows that 15 products' allowlists did transfer across products made by the same vendor, potentially because many same-vendor products share API endpoints.

Cameras were a key exception, as shown in Figure 8. Grouped by device type, this figure compares how MFAF distributions differ when a product's allowlist is applied to the product itself, as well as other products made by the same vendor. We found that allowlists created for cameras struggle the most when being applied on other same-vendor products, whose 25th quartiles drop from 0.97 to 0.43. Speakers' allowlists are in a similar situation; 25th quartiles drop from 0.99 to 0.50. One explanation is that the infrastructure required by a camera can be very different from that required by a switch or a light, making their allowlists harder to transfer to other products. The same observation applied to speakers.

In addition, some vendors may outsource some features to third-parties or purchase a company to acquire a product line, which often results in products with some unique behaviors. For example, most of Logitech's Harmony remote controllers' requests are sent to *"pubnub.com"*, which is a third-party company that provides real-time communication to computing applications and is critical to enable the away-from-home capability of Harmony [2]. Amazon Ring is another example, purchased by Amazon in 2018 [5], but keeps its own domains for communication.

6 Enforcing Allowlists in Lab

To test whether the allowlists generated in various circumstances impaired the functionality of devices, we deployed allowlists with various thresholds and host representations on nine devices in our lab. We intentionally exercised as many

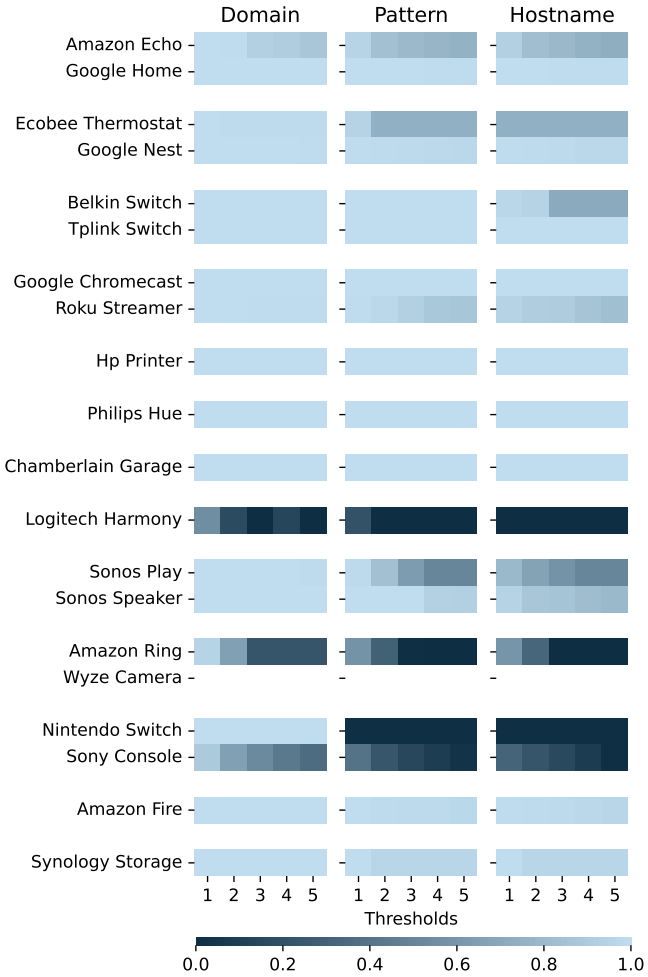

Figure 7: The MFAF of applying an allowlist to a particular product generated from the traffic of *other products from the same vendor*. Note that the Wyze camera row is blank because the dataset includes no other Wyze products.

functionalities as possible per device. We found that many of these capabilities seemed to work as intended, though we observed (and diagnosed) a few different failure modes. While we worried that rare behaviors (e.g., factory resets) unlikely to have been captured in IoT Inspector might prove problematic, this worry did not manifest in practice. Finally, the two-year gap between the collection of IoT Inspector and our experiments highlights the stability of allowlists.

6.1 Experiment Setup and Protocol

We selected the nine products most frequent (over 150 devices) in the IoT Inspector dataset. For highly similar products (e.g., Amazon Echo and Amazon Echo Dot), we only tested one. All devices were connected to a Jetson Nano through WiFi. The Jetson Nano, running Ubuntu 20.04, was used as a bridge to intercept devices' network traffic. We also deployed all generated allowlists on the Jetson Nano.

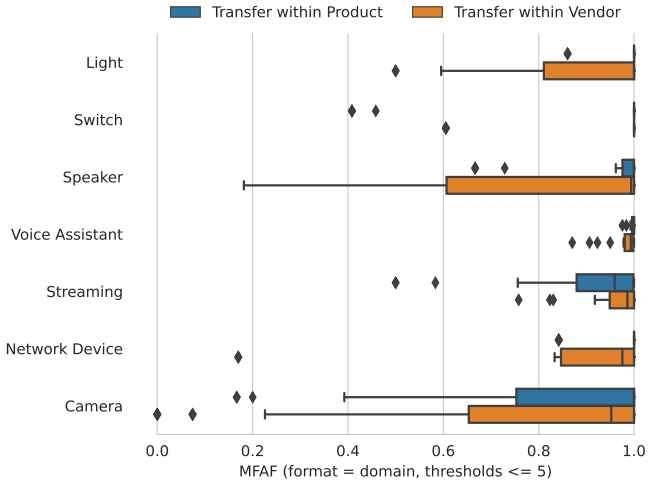

Figure 8: Distribution of vendor MFAF grouped by types. Each data point in the boxplot is the vendor MFAF of a product under the type category displayed on the y-axis. The boxplot uses domain-based allowlists with threshold ≤ 5.

Before running the tests, two researchers collaboratively determined each product's main functionalities and purposes using the product's manuals and companion app. We exclude functionalities that do not center on the device's main purpose (e.g., changing the device name). We also exclude functionalities that do not require an Internet connection (e.g., casting a webpage to a Google Chromecast), as confirmed in the lab.

We first conducted a measurement study to explore the expected behavior of each product, including rare operations (e.g., factory reset). The intention was to establish a baseline by documenting the behavior of each product's functionalities without allowlists. We then applied our allowlists to each product, proceeding from the most permissive to the most restrictive (by changing the thresholds defined in Section 5). Once an allowlist was deployed, we tried to exercise each of the product's documented functionalities. If any functionality failed, we repeated the experiment to confirm. The process ended either when all functionalities failed or the most restrictive allowlist was tested. Whenever possible, we tested each device from an external network (i.e., interacting with the companion app outside the lab's network) to maximize the product's Internet exposure. Only when devices cannot be controlled remotely (e.g., Google Chromecast) did we use the local network. We also used voice commands to control products and record the corresponding network traffic. Amazon Echo was used as the default receiver for voice commands since most devices we tested are compatible with Alexa.

Our approach differs from that of Mandalari et al. [24]. We evaluate allowlists for improving security, whereas Mandalari et al. studied blocklists for improving privacy. Further, Mandalari et al. obtained IoT destinations based on in-lab measurement, whereas our method used large-scale IoT Inspector data from real-world users.

6.2 Key Results

Of the 52 functionalities from the 9 devices we tested, 50 of them worked with at least one allowlist generated from the IoT Inspector dataset. 32 functionalities worked with a hostname-based allowlist, and 43 functionalities worked with pattern allowlists. In contrast, 50 of the 52 functionalities worked with domain-based allowlists. Table 2 details our results, including the maximum threshold at which the functionality continued to work and the size of the allowlist generated.

In some cases, compact allowlists were sufficient; 17 functionalities (33%) still worked when deploying an allowlist with fewer than 10 entries (Table 2). These functionalities were typically supported by one or more dedicated hostnames, making it possible to apply a hostname-based allowlist. For instance, Google Home only needs `www.google.com` to complete all functionalities other than media (e.g., playing music). The actual execution, such as controlling other devices or querying external information (e.g., weather), is handled through server-to-server communication.

For streaming media (audio or video), however, allowlists based on hostnames struggled. Of the 11 streaming functionalities we tested, 7 did not work with any hostname-based allowlists. Among these 7 functionalities, 4 started working after we switched to a pattern-based allowlist, and two others only worked with a domain-based allowlist. The final functionality, the Sonos Radio functionality, was a newly introduced feature. The endpoint it contacts, `sonos.radio`, was not observed in the two-year-old IoT Inspector dataset.

Despite this example, our results overall suggest that allowlists can be quite stable, at least for core functionalities. Among the nine products tested, three (Amazon Fire Stick, Belkin Wemo switch, and Sonos One) had functionalities seemingly impacted by the age of the IoT Inspector dataset. While the endpoint for the Belkin Wemo had changed in the intervening two years, severely impacting functionality when allowlists were deployed, the other two devices were affected in only minor ways. Most vendors were still using the same endpoints/APIs, making the allowlists fairly stable over time.

6.3 Reasons For Failures

To inform the design of future allowlists, we investigated and attribute the root cause of functionalities being impaired.

Content Delivery Networks (CDNs): Among the 20 functionalities that hostname-based allowlists break, the use of CDNs was directly responsible for six, with another four being affected indirectly (e.g., pausing music is not relevant if the music cannot be loaded in the first place). CDN hostnames commonly included seemingly random numbers or letters. As a result, allowlists based on exact hostname matches frequently blocked legitimate communication.

Our lab experiments verified that pattern-based allowlists often mitigate this failure. For example, when we attempted to play a playlist, Sonos One contacted `guc3-accesspoint-a-vr15.ap.spotify.com`, which was blocked by hostname-based allowlists, but allowed by pattern-based ones. We also observed cases where hostnames were too random to be clustered by DBSCAN, such as hostnames from CloudFront (e.g., `d924eisv4ltzs.cloudfront.net`). If the hostname is not included in the IoT Inspector dataset, then it is necessary to have a domain-based rule instead.

Load Balancing: As reasoned in Section 4, even with a large dataset like IoT Inspector, not all load-balancing endpoints will be observed. The Amazon Echo Dot in the lab experiments contacted `avs-alexa-14-na.amazon.com` for Alexa services, which was not observed in the dataset. However, as many other Alexa AVS endpoints were observed in the dataset, a successful pattern was extracted, solving the problem.

Dependencies: Some functionalities were impaired due to blocking another functionality. For example, as mentioned earlier, hostname-based allowlists often impaired music streaming, including for Sonos One. All other functionalities, such as changing the volume, were thus no longer accessible.

Connectivity checking was a common, and important, dependency. Many devices, such as the Google Home and Belkin Wemo switch, perform connectivity checking. If it fails, then the device refuses to take any further commands until it believes it is again online. Google Home uses `www.google.com` for most functionalities, but connectivity checking is via `connectivitycheck.gstatic.com`. Blocking the latter leads to a Google Home that refuses to take voice commands, telling the user it has connectivity issues.

API Changes: Rarely, vendors will change their API and infrastructure, invalidating allowlists generated from past data. This happened with the Belkin Wemo switch. When the IoT Inspector dataset was collected in 2019, Belkin Wemo plugs used `api.xbcs.net` for their API, which is confirmed in previous papers [23, 24]. However, Belkin recently added a new hostname, `deviceapis.xwemo.com` not observed in our dataset. In our lab experiment, the failure to connect to the new API endpoint put the device offline, impairing all functionalities. Ironically, the original API endpoint was still live, and the switch still contacted it from time to time.

New Services: In addition to changes to API endpoints, the vendor may also add new features. In our case, Sonos Radio is a new feature launched in 2020 [34]. In the lab, Sonos radio required a new domain, `sonos.radio`, for communications, but this new domain had not been observed in IoT Inspector.

Third-Party Services: We anticipated that integration with third-party services chosen by users could be another source of failures, though we did not observe this to be the case in our experiments. Popular integrations with YouTube, Netflix, or Spotify had been observed in IoT Inspector. However, users who prefer less popular services may encounter problems.

Device	Functionality	Hostname		Hostname Pattern		Domain	
		# of Allowed Hostnames	Threshold	# of Allowed Patterns	Threshold	# of Allowed Domains	Threshold
Amazon Echo Dot (N=576)	Productivity	× Load balancing		22	115	2	518
	Entertainment	× Load balancing		× CDNs		4	460
	Device control	× Load balancing		22	115	2	518
	Communication	× Load balancing		× CDNs		11	57
	Shopping	× Load balancing		22	115	2	518
	Skills	× Load balancing		1982	1	91	4
	Factory reset	× Load balancing		22	115	4	460
Amazon Fire Stick (N=136)	Built-in Alexa	77	13	74	13	6	54
	Streaming	10486	1	5481	1	17	27
	Download apps	210	5	187	5	33	13
	Factory reset	× API change		× API change		× API change	
Sonos One (N=261)	Radio	× New services		× New services		× New services	
	Streaming	× CDNs		119	3	4	52
	Change volume	× Interaction dependencies		119	3	4	52
	Pause	× Interaction dependencies		119	3	4	52
	Voice control (streaming)	× Load balancing		× Load balancing		62	3
	Voice control (volume)	× Load balancing		521	1	62	3
	Factory Reset	411	2	223	2	3	78
Google Home (N=490)	Media	× CDNs		151	3	2	392
	Control Chromecast	8	245	7	245	2	392
	Find your phone	8	245	7	245	2	392
	Manage tasks	8	245	7	245	2	392
	Control your home	8	245	7	245	2	392
	Plan	8	245	7	245	2	392
	Factory reset	26	49	25	49	2	392
Google Chromecast (N=288)	Watch Live TV	× CDNs		230	2	47	5
	Factory reset	13	115	5	201	2	201
Wyze Camera* (N=167)	Live streaming	77	2	68	2	4	100
	Event recording	77	2	68	2	4	100
	Motion Tagging	77	2	68	2	4	100
	Night vision	77	2	68	2	4	100
	2-way audio	77	2	68	2	4	100
	Sharing	77	2	68	2	4	100
	Rules	77	2	68	2	4	100
	On/Off	77	2	68	2	4	100
	Factory Reset	18	16	17	16	4	100
Philips Hue (N=295)	On/Off	1	236	1	236	1	236
	Brightness	1	236	1	236	1	236
	Voice control	1	236	1	236	1	236
	Timer	1	236	1	236	1	236
	Routine	1	236	1	236	1	236
	Factory reset	6	59	6	59	3	59
Belkin Wemo (N=220)	On/Off	× API change		× API changes		13	5
	Scheduling	× API changes		× API changes		13	5
	Auto-off	× API changes		× API changes		13	5
	Away mode	× API changes		× API changes		13	5
	Factory Reset	2	110	2	110	1	198
TP-Link Switch (N=88)	On/Off	3	26	3	26	1	70
	Scheduling	3	26	3	26	1	70
	Timer	3	26	3	26	1	70
	Away mode	3	26	3	26	1	70
	Factory Reset	3	26	3	26	1	70

Table 2: The highest tested thresholds (i.e., the strictest allowlist) and the number of allowed hostnames/patterns/domains that still keep the devices functioning. × [Failure Reason] means none of our generated allowlists can keep the functionality running and the following text explains why. To force the tested devices to use the Internet instead of the local network for functionality, the companion app is used on a smartphone connected to a different network by default, unless the functionality under test requires the smartphone to be on the same network (e.g., Google Chromecast). ***Wyze camera*** **is marked with a * as the device only works with some additional manually allowed IP addresses. The IP addresses are the main contributors to the Wyze Camera's traffic and can be easily observed, but no associated DNS lookup has ever been spotted.**

7 Related Work

Generating IoT network policies: Our work aims to understand how allowlists generalize across devices for various products by exploring the design space of allowlists through a large-scale IoT network dataset. Generating allowlists is merely a step in our exploration, instead of the goal of this paper. Our focus on allowlists makes the paper different from other works that focus on blocklists, which are easier to create and also primarily designed for privacy, not security [24]. As we care more about how generalizable an allowlist can be, creating allowlists based on observing a single device, as done in prior work [17, 20, 21], does not suffice. It also contrasts with proposals like Cisco's Manufacturer Usage Descriptions (MUD) [3], which asks IoT manufacturers to specify Internet hosts with which their IoT device should communicate. In contrast, our method requires no additional work from IoT vendors. Furthermore, our method also contrasts with systems like Bro [28] that focus on analyzing network flows at high speeds, forcing the user to decide what policies to implement.

Scale and realism of training data: The usage of the IoT Inspector dataset is also one key difference we have with other works, as it collects real-world network traffic from 5,439 devices from around the world [23]. Many existing approaches use supervised machine learning techniques to detect anomalous flows [9, 14, 18, 19, 26, 27, 31, 32, 35], but they typically rely on training and evaluation data from dozens of devices within the lab environment. For example, Dong et al. [13] trained neural networks to fingerprint IoT traffic based on a dataset of 10 IoT devices in the lab. Similarly, Mandalari et al. [24] identified blockable IoT-contacted hostnames based on an in-lab study of 31 devices. It is unclear whether these existing techniques would still be applicable on a larger IoT traffic dataset. In addition to academic work, some commercial products [1, 4] claim to generate rules from their vast customer's proprietary data, whereas our method generates rules based on an open, reproducible dataset.

In addition to the scale of our training data, our novelty also lies in the realism of the dataset. The dataset was collected from more than 5,000 volunteers from around the world [23], reflecting the behaviors of devices deployed under real-world human-device interactions. Much existing work, such as DeMarinis et al. [11] and even MUD [3], assumes that IoT devices have predictable patterns in the network traffic. Also, much work, including Dong et al. [13] and Mandalari et al. [24], is based on static snapshots of IoT device behaviors in the lab and could miss dynamic changes in devices behaviors, such as from firmware updates.

The potentially variable and complex behaviors of IoT devices call for techniques that dynamically build firewall-style rules. Techniques similar to ours include HANZO, which identifies devices and classifies network traffic from the home gateway [30]. Like much existing work, HANZO was evaluated in a lab setting, and it is unclear whether the method can capture the complex behaviors of real-world devices. Similarly, HanGuard instead constructs network policies by examining the traffic between IoT devices and mobile apps [12].

Alternate Approaches: Other work tries to protect compromised IoT devices using very different approaches to network security. For example, Martin et al. [25] protect IoT devices on private networks from external attackers by scrambling the port forwarding on the gateway. Acar et al. [6] proposes securing HTTP servers on IoT devices to prevent the lateral movement of malware. We instead examine communications between home IoT devices and hosts on the Internet.

8 Discussion & Conclusion

Home IoT devices have remained a significant attack surface, due to the prevalence of unpatched security vulnerabilities. Although relying on vendors to secure their devices through regular application of security updates certainly offers one level of defense, such an approach is incomplete, due to the fact that not all vendors regularly apply updates, and in some cases IoT devices can remain connected well past the time vendors are supporting them. This state of affairs means that some network-level defense is necessary, but determining precisely which traffic flows should be allowed from any particular device is surprisingly challenging, due to the significant heterogeneity of behavior across devices—even devices of exactly the same vendor and type. This heterogeneity makes it very challenging to derive general allowlists, which must account for diverse behavior while also maintaining a narrow attack surface.

This paper has explored this design space, evaluating how factors like the representation of a network endpoint (i.e., hostname, domain, AS), geographic location, and amount of observed traffic from the device can contribute to the construction of an allowlist that generalizes across devices of the same type or vendor. Our evaluation is promising, suggesting that general allowlists based on hostname patterns can be constructed based on a training sample of only about one or two dozen devices. We discovered, as well, that localizing the training data by region can be helpful, since devices do exhibit behaviors that are specific to their geography. Our evaluation of these constructed allowlists on real devices in an in-lab study—two years after the training set was collected—demonstrated that such allowlists generally work, reducing the attack surface without significantly impairing functionality. IoT devices that rely on cloud services often contact hostnames that include seemingly random numbers or patterns, making generalizability difficult and opening a security hole, since the allowlist hostname patterns we need to generate to capture these destinations can potentially be quite broad. Different strategies for generating and assigning these hostnames in the future could help manage this tradeoff.

References

[1] Bitdefender box,. https://www.bitdefender.com/box.

[2] Logitech powers smart home hub, remote control, and app using pubnub. https://www.pubnub.com/customers/logitech/.

[3] Manufacturer usage description specification. https://datatracker.ietf.org/doc/rfc8520/.

[4] Norton core router. https://us.norton.com/core.

[5] Ring (company) - wikipedia. https://en.wikipedia.org/wiki/Ring_(company).

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

[7] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE Symposium on Security and Privacy*, pages 1362–1380, 2019.

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

[9] Suman Sankar Bhunia and Mohan Gurusamy. Dynamic attack detection and mitigation in iot using sdn. In *2017 27th International telecommunication networks and applications conference (ITNAC)*, pages 1–6. IEEE, 2017.

[10] Ezra Caltum and Ory Segal. Sshowdown - exploitation of iot devices for launching mass-scale attack campaigns, 2016. https://tinyurl.com/3tbe358x.

[11] Nicholas DeMarinis and Rodrigo Fonseca. Toward usable network traffic policies for iot devices in consumer networks. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, pages 43–48, 2017.

[12] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A Gunter, Xiaoyong Zhou, and Michael Grace. 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*, pages 122–133, 2017.

[13] Shuaike Dong, Zhou Li, Di Tang, Jiongyi Chen, Menghan Sun, and Kehuan Zhang. Your smart home can't keep a secret: Towards automated fingerprinting of iot traffic. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 47–59, 2020.

[14] Tomer Golomb, Yisroel Mirsky, and Yuval Elovici. Ciota: Collaborative iot anomaly detection via blockchain. *arXiv preprint arXiv:1803.03807*, 2018.

[15] H. Guo and J. Heidemann. Iotsteed: Bot-side defense to iot-based ddos attacks (extended). USC/ISI Technical Report ISI-TR-738.

[16] Naman Gupta, Vinayak Naik, and Srishti Sengupta. A firewall for internet of things. In *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*, pages 411–412, 2017.

[17] Javid Habibi, Daniele Midi, Anand Mudgerikar, and Elisa Bertino. Heimdall: Mitigating the internet of insecure things. *IEEE Internet of Things Journal*, 4(4):968–978, 2017.

[18] Ibbad Hafeez, Aaron Yi Ding, Lauri Suomalainen, Alexey Kirichenko, and Sasu Tarkoma. Securebox: Toward safer and smarter iot networks. In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, pages 55–60, 2016.

[19] Ibbad Hafeez, Aaron Yi Ding, and Sasu Tarkoma. Ioturva: Securing device-to-device (d2d) communication in iot networks. In *Proceedings of the 12th Workshop on Challenged Networks*, pages 1–6, 2017.

[20] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Matthew Roughan, and Vijay Sivaraman. Clear as MUD: generating, validating and applying iot behavioral profiles. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, pages 8–14. ACM, 2018.

[21] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Theophilus A. Benson, Matthew Roughan, and Vijay Sivaraman. Verifying and monitoring iots network behavior using mud profiles. *IEEE Transactions on Dependable and Secure Computing*, 2020.

[22] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and analysis of hajime, a peer-to-peer iot botnet. In *Proc. NDSS*, 2019.

[23] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(2):1–21, 2020.

[24] Anna Maria Mandalari, Daniel J. Dubois, Roman Kolcun, Muhammad Talha Paracha, Hamed Haddadi, and David R. Choffnes. Blocking without breaking: Identification and mitigation of non-essential iot traffic. *PETS*, 2021.

[25] Vincentius Martin, Qiang Cao, and Theophilus Benson. Fending off iot-hunting attacks at home networks. In *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*, pages 67–72, 2017.

[26] Samuel Mergendahl, Devkishen Sisodia, Jun Li, and Hasan Cam. Source-end ddos defense in iot environments. In *Proceedings of the 2017 workshop on internet of things security and privacy*, pages 63–64, 2017.

[27] Mehdi Nobakht, Vijay Sivaraman, and Roksana Boreli. A host-based intrusion detection and mitigation framework for smart home iot using openflow. In *2016 11th International conference on availability, reliability and security (ARES)*, pages 147–156. IEEE, 2016.

[28] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[29] Anna Kornfeld Simpson, Franziska Roesner, and Tadayoshi Kohno. Securing vulnerable home iot devices with an in-hub security manager. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 551–556, 2017.

[30] Aman Singh, Shashank Murali, Lalka Rieger, Ruoyu Li, Stefan Hommes, Radu State, Gaston Ormazabal, and Henning Schulzrinne. Hanzo: Collaborative network defense for connected things. In *2018 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, pages 1–8. IEEE, 2018.

[31] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Characterizing and classifying iot traffic in smart cities and campuses. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 559–564. IEEE, 2017.

[32] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Vijay Sivaraman, and Arun Vishwanath. Low-cost flow-based security solutions for smart-home iot devices. In *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2016.

[33] Statista. Do you own smart home devices - i.e. devices that you can control via a smartphone / an internet connection?, Jul 2021. https://www.statista.com/forecasts/1097129/smart-home-device-ownership-in-selected-countries.

[34] Chris Welch. Sonos launches its own streaming radio service, 2020. https://www.theverge.com/2020/4/21/21228460/sonos-radio-announced-features-streaming-music-date-price.

[35] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. 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*, pages 1–7, 2015.

[36] Nan Zhang, Soteris Demetriou, Xianghang Mi, Wenrui Diao, Kan Yuan, Peiyuan Zong, Feng Qian, XiaoFeng Wang, Kai Chen, Yuan Tian, et al. Understanding iot security through the data crystal ball: Where we are now and where we are going to be. *arXiv:1703.09809*, 2017.

A Functionalities Tested in the Lab

Functionality	Description
Productivity	"Alexa, what's the time?"
Entertainment	"Alexa, play music."
Device control	"Alexa, turn on/off all Hue lights."
Communication	"Alexa, Call Alice."
Shopping	"Alexa, search for dog toys."
Skills	"Alexa, play thunderstorm sounds."
Change volume	Change the volume of the Echo Dot.
Factory Reset	Reset and initialize the device.

Table 3: Functionality descriptions of **Amazon Echo Dot**.

Functionality	Description
On/Off	Turn lights on/off.
Brightness	Change the brightness level of a light.
Voice control	Control the device (on/off) using Alexa.
Timer	Activate a timer that would turn lights on/off in one minute.
Routine	Change time to fit for the experiment
Factory Reset	Reset and initialize the device.

Table 4: Functionality descriptions of **Philips Hue**.

Functionality	Description
Built-in Alexa	"Alexa, find action movies"
Streaming	Streaming a movie that is free (with ads).
Download apps	Download Spotify in the App Market on Amazon Fire.
Factory Reset	Reset and initialize the device.

Table 5: Functionality descriptions of **Amazon Fire Stick**.

Functionality	Description
Media	"Google, play some music."
Control Chromecast	"Hey Google, play Youtube on the Hallway TV"
Find your phone	"Hey Google, find my phone"
Manage tasks	"Hey Google, set a timer for 1 min."
Control your home	"Hey Google, turn on the wemo plug."
Plan	"Hey Google, how's the weather?"
Change volume	
Factory Reset	Reset and initialize the device.

Table 6: Functionality descriptions of **Google Home**.

Functionality	Description
Watch Live TV	"Hey Google, play Youtube on the Hallway TV"
Cast tab to TV	
Cast screen to TV	
Cast media to TV	
Factory Reset	Reset and initialize the device.

Table 7: Functionality descriptions of **Google Chromecast**.

Functionality	Description
Live streaming	Watch the live stream from the camera.
Event recording	Record the live stream.
Motion Tagging	Tag motions.
Night vision	Switch the camera to night vision.
2-way audio	Two-way communication through the camera.
Sharing	Share a video clip to other family members.
Rules	
On	Turn on the camera.
Off	Turn off the camera.
Factory Reset	Reset and initialize the device.

Table 8: Functionality descriptions of **Wyze Camera**.

Functionality	Description
On	Turn on the plug.
Off	Turn off the plug.
Scheduling	Schedule the plug to turn on/off at a specific time.
Auto-off	Set up a one-minute countdown to turn off the plug.
Away mode	Activate the away mode.
Factory Reset	Reset and initialize the device.

Table 9: Functionality descriptions of **Belkin Wemo Plug**.

Functionality	Description
On	Turn on the plug.
Off	Turn off the plug.
Scheduling	Schedule the plug to turn on/off at a specific time.
Timer	Set up a one-minute countdown to turn off the plug.
Away mode	Activate the away mode.
Factory Reset	Reset and initialize the device.

Table 10: Functionality descriptions of **TP-Link Plug**.

Functionality	Description
Radio	Play Sonos Radio.
Streaming	Play songs from Spotify.
Change volume	Change volume.
Pause	Pause the music.
Play songs from phone	Play a song from a phone under the same network.
Voice control (streaming)	Using Alexa to play songs.
Voice control (volume)	Using Alexa to change volume.
Playlists	Edit playlists.
Factory Reset	Reset and initialize the device.

Table 11: Functionality descriptions of **Sonos One**.

B Regression Tables of Regionalization

Factor	β	SE	*t*	*p*
(Intercept)	0.828	0.004	194.915	**<.001**
Train Region: B	-0.025	0.001	-19.322	**<.001**
Train Region: C	-0.036	0.001	-26.541	**<.001**
Test Region: B	-0.041	0.002	-22.423	**<.001**
Test Region: C	-0.091	0.004	-24.806	**<.001**
Product: Belkin Switch	-0.363	0.002	-170.291	**<.001**
Product: Google Chromecast	0.086	0.002	51.714	**<.001**
Product: Google Home	0.089	0.001	65.493	**<.001**
Product: Philips Hue	0.074	0.002	43.744	**<.001**
Product: Sonos Speaker	-0.321	0.002	-194.669	**<.001**
log(Train Sample Size)	0.008	<.001	18.014	**<.001**
Train Region: B * Test Region: B	0.060	0.003	23.276	**<.001**
Train Region: B * Test Region: C	0.063	0.005	12.131	**<.001**
Train Region: C * Test Region: B	0.054	0.003	21.339	**<.001**
Train Region: C * Test Region: C	0.128	0.007	19.644	**<.001**

(a) Hostname-based allowlists

Factor	β	SE	*t*	*p*
(Intercept)	0.792	0.004	213.467	**<.001**
Train Region: B	-0.031	0.001	-25.069	**<.001**
Train Region: C	-0.032	0.001	-25.500	**<.001**
Test Region: B	-0.036	0.002	-21.423	**<.001**
Test Region: C	-0.100	0.003	-29.007	**<.001**
Product: Belkin Switch	-0.044	0.002	-22.248	**<.001**
Product: Google Chromecast	0.063	0.002	40.579	**<.001**
Product: Google Home	0.066	0.001	51.954	**<.001**
Product: Philips Hue	0.048	0.002	30.074	**<.001**
Product: Sonos Speaker	-0.187	0.002	-121.330	**<.001**
log(Train Sample Size)	0.008	0.000	18.412	**<.001**
Train Region: B * Test Region: B	0.074	0.002	30.675	**<.001**
Train Region: B * Test Region: C	0.055	0.005	11.369	**<.001**
Train Region: C * Test Region: B	0.064	0.002	27.024	**<.001**
Train Region: C * Test Region: C	0.121	0.006	19.944	**<.001**

(b) Hostname-pattern-based allowlists

Factor	β	SE	*t*	*p*
(Intercept)	0.915	0.004	225.965	**<.001**
Train Region: C	-0.034	0.001	-30.047	**<.001**
Train Region: B	-0.013	0.001	-11.484	**<.001**
Test Region: C	-0.053	0.003	-17.047	**<.001**
Test Region: B	-0.021	0.002	-13.376	**<.001**
Product: Belkin Switch	-0.093	0.002	-54.276	**<.001**
Product: Google Chromecast	-0.002	0.001	-1.303	0.193
Product: Google Home	0.014	0.001	12.889	**<.001**
Product: Philips Hue	0.044	0.001	31.927	**<.001**
Product: Sonos Speaker	-0.097	0.001	-68.704	**<.001**
log(Train Sample Size)	0.004	<.001	11.531	**<.001**
Train Region: C * Test Region: C	0.092	0.006	16.720	**<.001**
Train Region: B * Test Region: C	0.036	0.004	8.231	**<.001**
Train Region: C * Test Region: B	0.044	0.002	20.333	**<.001**
Train Region: B * Test Region: B	0.022	0.002	10.072	**<.001**

(c) Domain-based allowlists

Table 12: (Continued) Linear regression modeling the impact on transferability of the geographic region in which devices were located, whether the train and test regions were the same (* represents an interaction term), the amount of training data, and the product. As the baseline for categorical variables, we use the largest category: *A* for region and *Amazon Echo* for product.

C Additional Figures

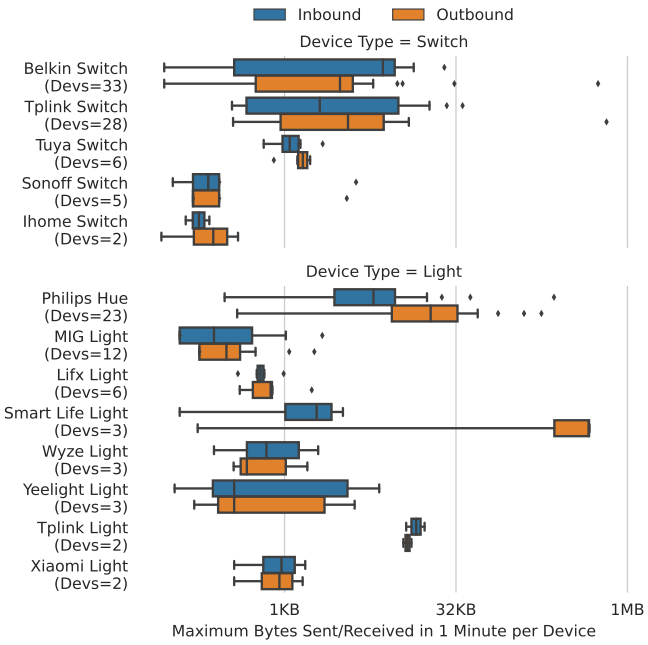

Figure 9: The maximum bytes sent or received in one minute by Switches and Lights, categorized by products.

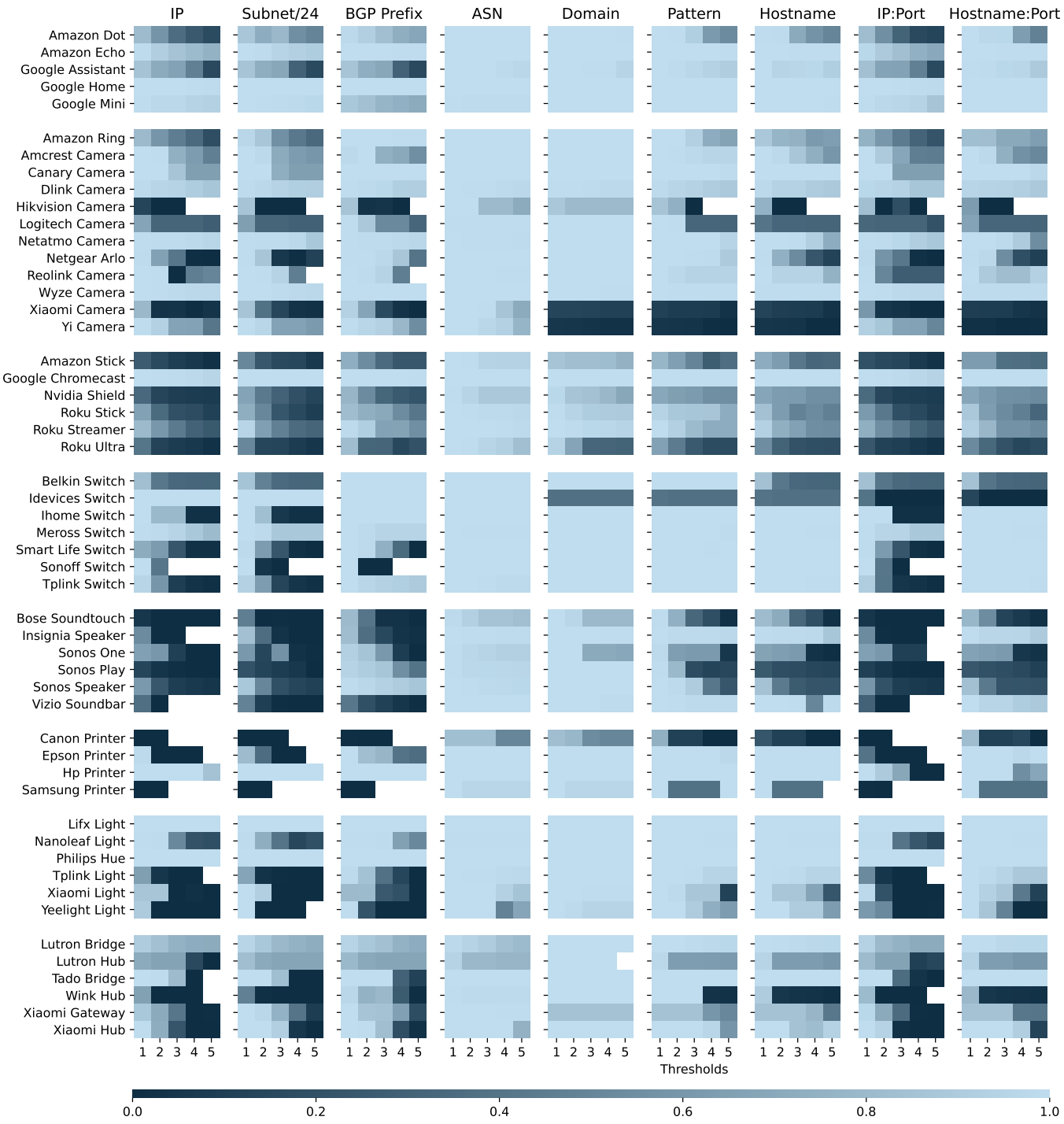

Figure 10: Transferability of device network behavior using different allowlist formats (columns). Rows represent the source product for generating the allowlists, and the color indicates the proportion of flows allowed when creating an allowlist using a threshold of between 1 and 5s (sub-columns) and applying it to devices of **the same product**.

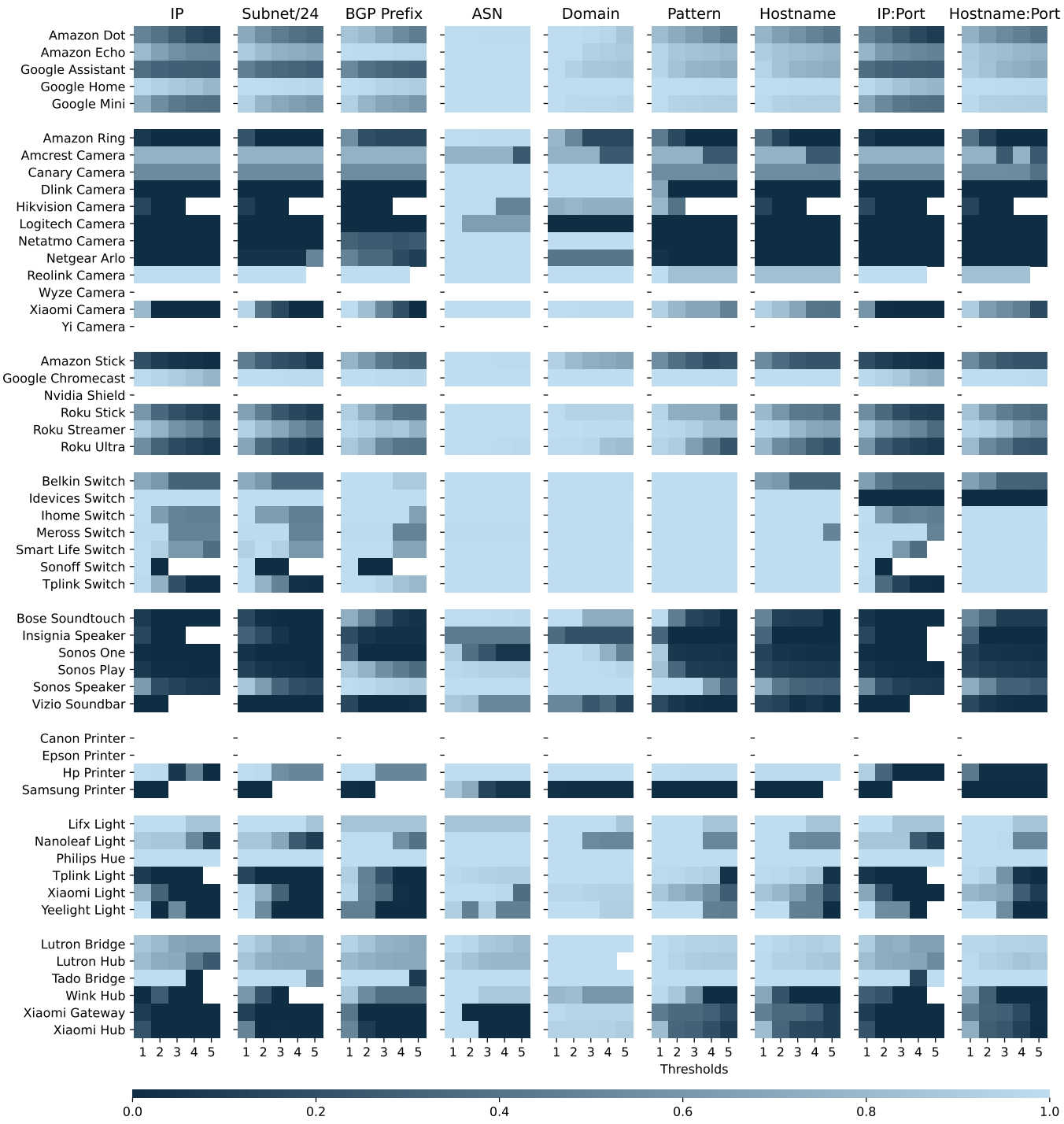

Figure 11: Transferability of device network behavior using different allowlist formats (columns). Rows represent the source product for generating the allowlists, and the color indicates the proportion of flows allowed when creating an allowlist using a threshold of between 1 and 5s (sub-columns) and applying it to devices of **the same vendor**.

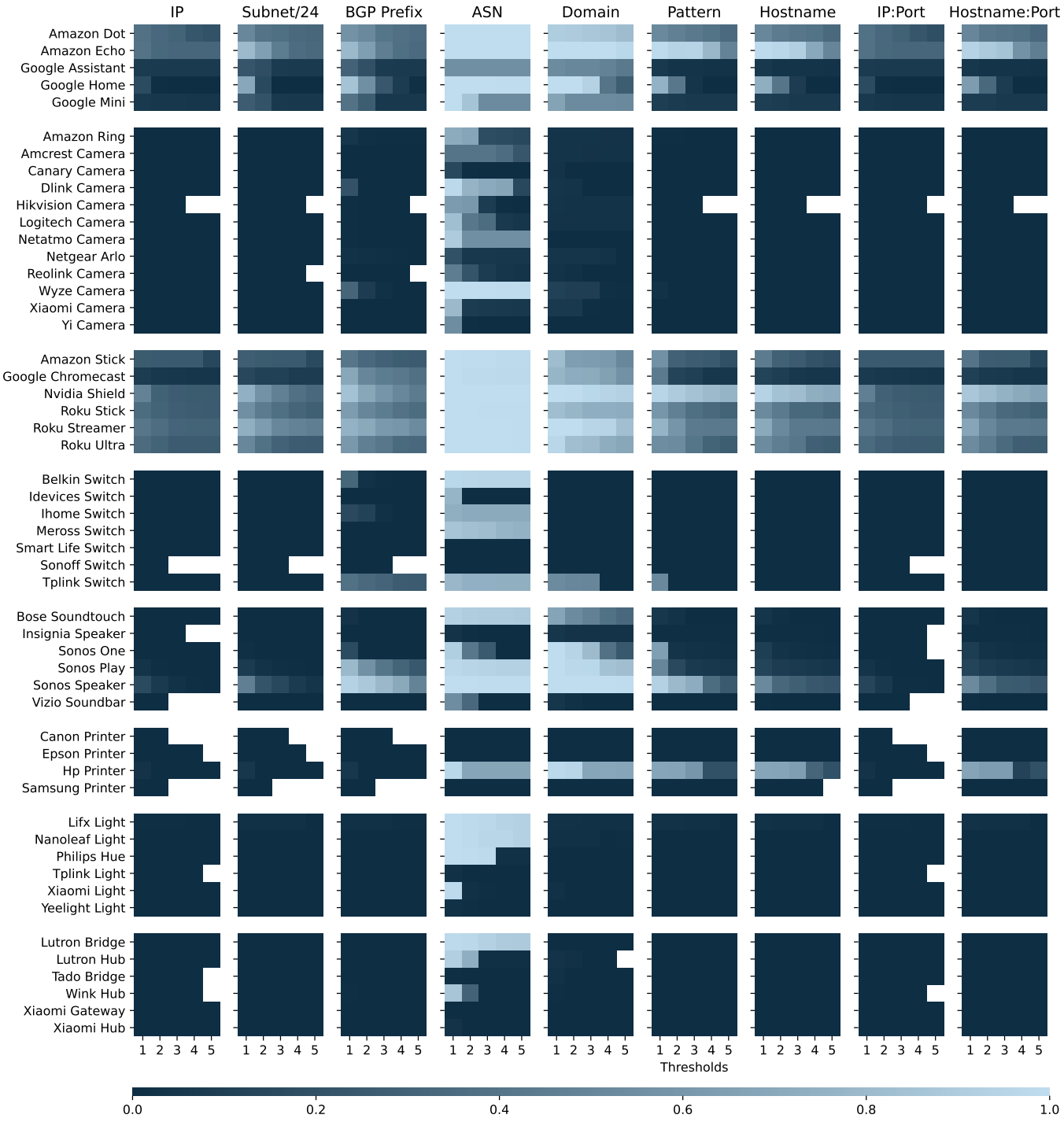

Figure 12: Transferability of device network behavior using different allowlist formats (columns). Rows represent the source product for generating the allowlists, and the color indicates the proportion of flows allowed when creating an allowlist using a threshold of between 1 and 5s (sub-columns) and applying it to devices of **the same type**.