IoTSTEED: Bot-side Defense to IoT-based DDoS Attacks (Extended)

USC/ISI Technical Report ISI-TR-738 June 2020

Hang Guo
hangguo@isi.edu
USC/ISI

John Heidemann
johnh@isi.edu
USC/ISI

ABSTRACT

We propose IoTSTEED, a system running in edge routers to defend against Distributed Denial-of-Service (DDoS) attacks launched from compromised Internet-of-Things (IoT) devices. IoTSTEED watches traffic that leaves and enters the home network, *detecting* IoT devices at home, *learning* the benign servers they talk to, and *filtering* their traffic to other servers as a potential DDoS attack. We validate IoTSTEED's accuracy and false positives (FPs) at detecting devices, learning servers and filtering traffic with replay of 10 days of benign traffic captured from an IoT access network. We show IoTSTEED correctly detects all 14 IoT and 6 non-IoT devices in this network (100% accuracy) and maintains low false-positive rates when learning the servers IoT devices talk to (flagging 2% benign servers as suspicious) and filtering IoT traffic (dropping only 0.45% benign packets). We validate IoTSTEED's true positives (TPs) and false negatives (FNs) in filtering attack traffic with replay of real-world DDoS traffic. Our experiments show IoTSTEED mitigates all typical attacks, regardless of the attacks' traffic types, attacking devices and victims; an intelligent adversary can design to avoid detection in a few cases, but at the cost of a weaker attack. Lastly, we deploy IoTSTEED in NAT router of an IoT access network for 10 days, showing reasonable resource usage and verifying our testbed experiments for accuracy and learning in practice.

1 INTRODUCTION

There is an increasing concern about the security threats that Internet-of-Things (IoT) devices, such as Internet-enabled light bulbs and cameras, raise for the Internet ecosystem. The massive number of IoT devices, together with their often inadequate security [10, 11, 66] and even unpatchabilities [68], make them attractive targets for compromises. One flagrant example is that compromised IoT devices (as known as"bots") , besides risking leaking sensitive information like audio [12, 25] and video recordings [67, 81], could be used to mount large-scale Distributed Denial-of-Service (DDoS) attacks and significantly damage Internet security. In 2016, over 100k compromised IoT devices , compromised by IoT malwares Mirai [47], launched a series of record-breaking DDoS attacks, including a 620 Gb/s attack against krebsonsecurity.com (2016-09-20) [44] and 1 Tb/s attacks against cloud-provider OVH (2016-09-23) [60] and DNS-provider Dyn (2016-10-21) [17].

A naive way to defend IoT-based DDoS attacks is to make all IoT devices secure. However, IoT manufacturers are not incentivized to produce more secure and likely more expensive products, because they may see less sales from price-sensitive customers [77, 79]. While there are emerging legislative efforts for enforcing IoT security, such as the IoT Cybersecurity Improvement Act of 2019 [42] requiring minimum security for government-purchased IoT products, these proposals are not yet laws.

Another option is to mitigate IoT-based DDoS attack at victims as widely explored [1, 4, 8, 35, 39–41, 43, 46, 57, 62–65, 75, 76, 78, 80]. However, due to the large number of IoT devices (5.8 billion in 2020 [19]), and the resulting high volume of attack traffic (at most 1 Tb/s for one attack as of 2016), target-side filtering is very costly and is only possible for the largest operators today. (For example, Akamai estimated costs of one defense as millions of dollars [5]).

In this paper, we advocate the third option: defending IoT-based DDoS attacks at bot-side. The main advantage of bot-side defense is much lower volumes of attack traffic compared to at the victim. As a result, bot-side defense is less costly and more likely to cope with future growth in attack volume. (Bot-side defense also inherently supports incremental deployment: when deployed in only a fraction of access networks in the Internet, it still mitigate DDoS traffic at victim proportionally.)

Our first contribution is to propose *IoTSTEED* (IoT bot-Side Traffic-Endpoint-basEd Defense), a system runs in edge routers to defend against DDoS attacks launched from compromised IoT devices. IoTSTEED watches traffic that leaves and enters the home network, *detecting* IoT devices at home (§2.1), *learning* the benign servers they talk to (§2.1), and *filtering* their traffic to other servers as a potential DDoS attack (§2.3).

Our second contribution is to validate IoTSTEED's correctness with with replay of off-line traffic capture (§3). We validate IoTSTEED's accuracy and false positives (FPs) in detecting devices, learning server and filtering traffic with replay of 10 days of benign traffic captured from an IoT access network (§3.1). We show IoTSTEED correctly detects all 14 IoT and 6 non-IoT devices in this network (100% accuracy) and maintains low false-positive rates when learning the servers IoT devices talk to (flagging 2% benign servers as suspicious) and filtering IoT traffic (dropping only 0.45% benign packets). We validate IoTSTEED's true positives (TPs) and false negatives (FNs) in filtering attack traffic with replay of real-world DDoS traffic (§3.2). Our experiments show IoTSTEED could mitigate all typical attacks, regardless of the attacks' traffic types, attacking devices and victims. An intelligent adversary can design to avoid detection in a few cases, but at the cost of a weaker attack.

Our third contribution is to deploy IoTSTEED in NAT router of an IoT access network for 10 days (§4). We show IoTSTEED runs well on a commodity router: memory usage is small (4% of 512MB) and the router forwards traffic at full uplink rates. We confirm IoTSTEED's accuracy, FPs, TPs and FNs in detecting devices, learning servers and filtering traffic during on-line router deployment is similar to what we report in off-line trace-replay validation.

Manufacturer	Device Type (Model)	Alias
Amcrest	IP Camera (IP2M-841)	Amcrest_Cam
Belkin	Smart Plug (Wemo Mini)	Belkin_Plug
Dyson	Air Purifier (Pure Cool Link)	Dyson_Purifier
D-Link	IP Camera (DCS-934L)	D-Link_Cam
Foscam	IP Camera (FI8910W)	Foscam_Cam
Foscam	IP Camera (R2C)	Foscam_Cam2
HP	Wireless Printer (Envy 4500)	HP_Printer
Samsung	IP Camera (SNH-P6410BN)	Samsung_Cam
Philips	Light Bulb (Hue A19 Kit)	Philips_Bulb
TP-Link	Smart Plug (HS100)	TPLink_Plug
TP-Link	Light Bulb (LB110)	TPLink_Bulb
Tenvis	IP Camera (WH-TH661)	Tenvis_Cam
Wyze	IP Camera (WYZEC2)	Wyze_Cam
Wansview	IP Camera (633GBU)	Wansview_Cam

Table 1: 14 IoT devices We Own

(We have released IoTSTEED's source code [28] and the 10-day benign IoT traffic capture from validation [27].)

2 METHODOLOGY

IoTSTEED follows the observation that IoT devices usually talk to a small number of benign servers (from our prior work [29, 30]). By whitelisting these benign servers, it can mitigate suspicious IoT traffic to all other servers.

IoTSTEED examines packets entering and leaving an IoT access network from its edge router, detecting IoT devices in this network (§2.1), learning benign servers these IoT devices talk to (§2.2) and filtering their traffic to other servers as a potential DDOS attack (§2.3). IoTSTEED is thus an instance of network anomaly detection which learns and profiles benign traffic and identifies traffic deviating from benign profiles as malicious. Prior work has shown that network anomaly detection does not work well in practice because profiling highly-variable real-world network traffic is hard [72]. We show that since IoT traffic is relatively simple, we could at least profile benign IoT traffic endpoints (§3.1).

IoTSTEED thus focuses on single-purpose IoT devices, such as smart plugs and cameras, that talk to a small amount of server names. IoTSTEED does not work with multiple-purpose IoT devices, such as smart TV, that could talk to hundreds of server names by installing new applications.

IoTSTEED currently handles IPv4 traffic since our test home network is v4-only (§3 and §4); adding IPv6 support should be straightforward. We believe that our results of defending IPv4 attacks (§3 and §4) prove the effectiveness of our system and we leave defending IPv6 attacks as future work.

2.1 Device Detection

IoTSTEED first detects IoT devices in the access network where it runs. It later learns the benign servers these IoT devices talk to (§2.2) and filters IoT traffic to other servers (§2.3).

2.1.1 Overview. IoTSTEED's device detection follows the observation that many IoT manufacturers only produce IoT devices. Therefore it can detect IoT devices by mapping their MAC addresses to their manufacturers and identifying known IoT manufacturer names (§2.1.2 and §2.1.3). Since IoTSTEED's detection mis-classifies non-IoT devices made by IoT manufacturers as IoT, it corrects these mis-classifications by identifying the large number of server names these non-IoT devices talk to (§2.1.4).

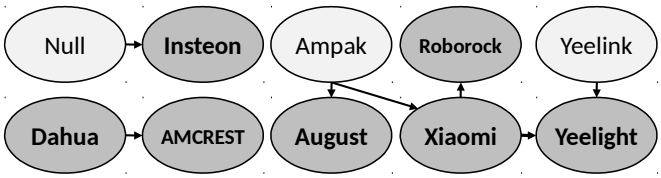

Figure 1: Part of the Directed Graph that Stores Known IoT Manufacturers and Relatives (Dark and Light Circles)

2.1.2 Collect IoT Manufacturer Names. To detect IoT devices by comparing their MAC-inferred manufacturers with known IoT manufacturers, we need to collect a list of IoT manufacturer names. However, knowing IoT manufacturers is not enough. Some IoT MAC addresses (about one third of 185 we examine in next paragraph and about one third of 522 that IoT inspector examines in [36]) get mapped to organizations related to the actual IoT manufacturers such as parts makers, original equipment manufacturers (OEMs) and parent companies of the IoT manufacturers. We call these "manufacturer-relatives" or simply "relatives", and collect relatives for each known IoT manufacturer. When an IoT MAC address gets mapped to a relative, we can narrow down this IoT device's potential manufacturers to a list of manufacturers related to this relative.

We collect a list of IoT manufacturers and relatives by first collecting a list of IoT MAC addresses with ground truth manufacturer names. We find 185 IoT MAC addresses from 67 IoT manufacturers based on devices we own (Table 1), public IoT traffic capture [2, 31, 70], and Google image searches (for example, we search "smart plug MAC address" for MAC addresses printed on bottom of smart plugs). We then find relatives for these 67 IoT manufacturers by looking up these 185 MAC addresses with a MAC-to-vendor mapping library [49] and identifying lookup results different from ground truth manufacturer names as relatives. (We ensure the first three octets of our 185 IoT MAC addresses, which uniquely identify vendors, are all distinct.) We show 67 of these 185 MAC addresses (36%) get mapped to 45 distinct relatives. As a result, we obtain 67 IoT manufacturers and 45 relatives.

To expedite finding potential manufacturers for IoT devices whose MAC addresses get mapped to relatives, we store the relationship between known IoT manufacturers and relatives in a directed graph. We store each manufacturer and its relatives as vertices in this graph and connect them with an edge pointing to this manufacturer vertex from its relative vertices. The resulting graph (part of which is shown in Figure 1) allows us to identify all manufacturers related to a relative by identifying all manufacturer vertices reachable from this relative's vertex.

We handle two edge cases in building this direct graph. For IoT manufacturers that are also relatives (such as IP camera manufacturers Dahua who also OEMs for other IP camera makers like

Amcrest [37]), we label them as IoT manufacturer in our graph (see "Dahua" manufacturer vertex in Figure 1). and infer they are also relatives from the fact that their vertices point to other manufacturer vertices in our graph. For IoT manufacturers who use MAC addresses that cannot be mapped to any organizations, or use private MAC addresses,we add a special relative ("null" or "private", see Figure 1).

Our IoT detection risks being incomplete because our knowledge of IoT manufacturers and their relatives is limited. In principle, we could scale up by crowd-sourcing IoT MAC addresses with ground truth manufacturer names, as shown in [36].

2.1.3 Detect IoT Devices by MAC Lookup. We next detect IoT devices and infer their potential manufacturers by looking up their MAC addresses and matching the lookup results with our knowledge of IoT manufacturers and relatives.

IoTSTEED detects IoT devices by examining the MAC addresses of every observed packet, looking up these MAC addresses with MAC-to-vendor mapping library [49] and identify those whose lookup results match certain vertices in our graph. (IoTSTEED classifies the rest MAC addresses as non-IoT devices.) IoTSTEED considers a MAC lookup result matching a vertex if this vertex's manufacturer (or relative) name is a substring of this lookup result (regardless of case) and every English word in this vertex show up in this lookup result. (For example, relative name "Physical Graph" matches lookup result "physical graph corp" because the former is a substring of the latter and both words "Physical" and "Graph" show up in the lookup result. In comparison, manufacturer name "Ring" does not match MAC lookup result "Murata Manufacturing Co. Ltd" because the word "Ring", despite being a substring of the result, does not exist in the result.)

When detecting a new IoT device, IoTSTEED infers its manufacturer (or a list of potential manufacturers) by finding all manufacturer vertices (directly or indirectly) reachable from the vertex found by the MAC address. For example, if an IoT device's MAC address gets mapped to parts maker "Ampak" in Figure 1, its potential manufacturers include "August" and "Xiaomi" who use parts from Ampak and "Roborock" and "Yeelight" who partner with Xiaomi.

2.1.4 Correct Potential Mis-classifications. IoTSTEED identifies non-IoT devices mis-classified as IoT (due to some IoT manufacturers also produce non-IoT devices) by looking for detected IoT devices that talk to excessive number of server names. The rationale is that we find non-IoT devices usually talk to more server names than IoT devices do (Figure 2). Specifically, if any IoT device DNS queries more than T_{svr} distinct servers names, IoTSTEED re-classifies it as non-IoT. We set T_{svr} as 70 based on examining 10-day operational traffic from 60 IoT devices and 6 non-IoT devices (we measure our 14 IoT and 6 non-IoT devices, as in Table 1, and use public traffic pcaps for the remaining 46 IoT devices [2, 31, 70]). As in Figure 2, we find these 60 IoT devices each queries at most 15 distinct server names (in average 5) while these 6 non-IoT devices each queries at least 128 distinct server names (in average 451) in 10 days.

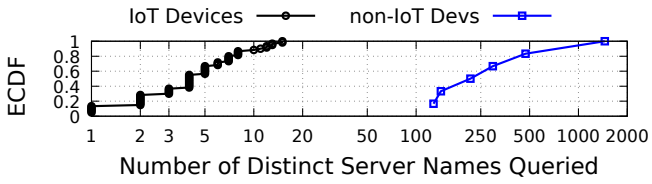

Figure 2: ECDF for Number of Distinct Server Names Queried by 60 IoT and 6 non-IoT Devices in 10 Days

2.2 Server Learning

IoTSTEED next learns benign servers detected IoT devices talk to (*IoT servers*). Knowing IoT servers enables it to filter IoT traffic to other servers (*non-IoT servers*) as a potential DDoS attack in §2.3.

2.2.1 Overview. IoTSTEED learns IoT servers from all servers detected IoT devices talk to in two rounds: server bootstrapping (§2.2.3) and expansion (§2.2.4). (IoTSTEED maintains a separate IoT server list for each IoT device.) IoTSTEED also whitelists a short list of server IPs that are always considered benign: Google's public DNS revolvers (8.8.8.8 and 8.8.4.4) that are often visited by IoT devices, public IPs of the NAT router where IoTSTEED runs (since we find IoT devices sometimes talk to router's public IPs) and public IPs of the mobile phone used to remote access IoT devices.

2.2.2 Server Identification. To learn IoT servers, IoTSTEED identifies servers by either their DNS names or IP addresses.

IoTSTEED identifies servers mainly by their DNS names because we find server names to be relatively stable over time while server IPs could change. We find some devices could visit servers directly by IPs without preceding DNS queries (such as Google's public DNS resolvers 8.8.8.8) and as a result, IoTSTEED identifies these servers by their IPs (called "IP-accessed servers" hereafter). (We call servers visited with preceding DNS queries "name-accessed servers".)

Since IoTSTEED mainly identifies servers by DNS names but sees server IPs in traffic, it tracks server name-to-IP mappings based on DNS resolutions observed from IoT devices. Specifically, IoTSTEED tracks a list of server names each IoT device talks to based on server names they queried using type A, AAAA and CNAME DNS requests. It then extracts server IPs and canonical names for these server names from corresponding type A and CNAME DNS replies. (IoTSTEED does not track AAAA DNS replies because it currently ignores non-DNS IPv6 traffic.)

2.2.3 First-Round Learning: Server Bootstrapping. IoTSTEED bootstraps a list of IoT servers for every IoT device by classifying all servers they DNS query (for name-accessed servers) or directly visit (for IP-accessed servers) shortly after most recent bootup as benign. The rationale is that we trust recently-bootup devices to be uncompromised and only talking to benign servers because IoT malwares usually do not sustain device reboot [3, 20, 22, 52] and re-infections take time (considering that many malware randomly scan for infection [3, 21, 52]).

Specifically, after IoTSTEED detects a new IoT device D (§2.1), it first estimates D's most recent bootup time (T_{bt}^{D}) with the timestamp of the first packet observed from D and then classifies all servers

that D DNS queries or directly visits between $[T_{bt}^D, T_{bt}^D + T_{sp})$ as benign, where T_{sp} is the duration of server bootstrapping.

IoTSTEED's estimation of T_{bt}^D holds intuitively if D is newly-acquired and first boot up after IoTSTEED starts. If D is an existing device, we require the owner to reboot it before starting IoTSTEED so that IoTSTEED could correctly estimate T_{bt}^D.

We experimentally set T_{sp} as two hours for name-accessed servers (annotated as T_{sp}^{abn}). By booting up our 14 IoT devices and observing them for 10 days, we find that they talk to most (75% or 67) of their 89 name-accessed IoT servers (blue bars in Figure 3) within the first two hours after bootup — bootstrapping behavior.

Similarly, we set T_{sp} as 120 hours for IP-accessed servers (T_{sp}^{abi}) because we find 10 of our 14 IoT devices (all except Foscam_Cam, Amcrest_Cam, Belkin_Cam and D-Link_Cam) talk to most (91% or 30) of their 33 IP-accessed IoT servers (gray bars in Figure 3) within the first 120 hours after bootup (green area in Figure 4).

The four IoT devices remaining (Foscam_Cam, Amcrest_Cam, Belkin_Cam and D-Link_Cam) show no such bootstrapping behavior and instead keep visiting new IP-accessed servers even after bootstrapping period (white area in Figure 4). Our server learning cannot handle them: bootstrapping-based first round only covers part of their IP-accessed IoT servers and server-name-based second round does not apply to IP-accessed servers. We choose to *not* filter traffic between these devices and their IP-accessed IoT servers (called "turn off IP-accessing filtering") to reduce potential false positives (FPs) in traffic filtering. (See §2.3 for details.)

We show these four devices' lack of bootstrapping behavior in visiting IP-accessed servers is mainly an artifact of UPnP service in our router. We find three of them (all except Belkin_Plug) set up static port mappings in our routers via UPnP. (We confirm Foscam_Cam uses UPnP for remote device accessing but are not certain about other devices.) As a side effect, they get unsolicited packets from a large number of remote IPs (574 or 98% of their 586 IP-accessed IoT servers) such as scanners from internet-census.org and shodan.io. By responding to these unsolicited packets, these three devices appear constantly talking to new IP-accessed servers. We support our hypothesis that UPnP causes lack of bootstrapping behavior by showing that without UPnP, these three devices do show bootstrapping behavior in a similar 10-day experiment (§4). For the remaining one device (Belkin_Plug) that does not use UPnP, its 22 IP-accessed IoT servers are mostly STUN servers for NAT traversal (73% or 16). One explanation for Belkin_Plug's lack of bootstrapping behavior is that it keeps connecting to different STUN servers IPs for NAT relay services.

UPnP service also explains the large number of IP-accessed IoT servers (641, as in Figure 3) our IoT devices talk to. The three devices (Foscam_Cam, Amcrest_Cam and D-Link_Cam) that contributes to almost all (91%, 586) of these 641 IP-accessed IoT servers all set up static port mappings via UPnP. We have shown their 586 IP-accessed IoT servers is mainly an artifact of them responding to unsolicited probes from remote IPs (98%, 574). To support our hypothesis that UPnP service inflates IP-accessed server count, we show that without UPNP, our 14 devices talk to only 69 IP-accessed IoT servers (9× less than 641 servers with UPnP) in 10 days (§4).

While we used public IoT traces [2, 31, 70] for collecting IoT manufacturer names (§2.1.2) and setting T_{svr} values (§2.1.4), we

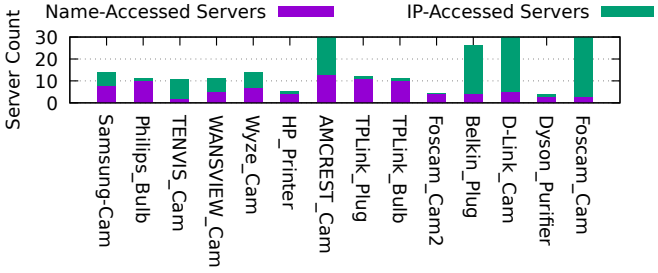

Figure 3: Distinct Name-accessed and IP-accessed IoT Servers Our Devices Visit Within 10 Days of Bootup (Amcrest_Cam, D-Link_Cam and Foscam_Cam Visit 117, 328 and 141 Servers but Get Cropped for Displaying).

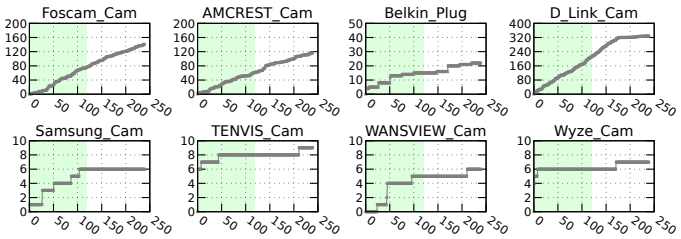

Figure 4: Distinct IP-accessed IoT Servers Our Devices Talk to Per Hour Within 10 Days of Bootup (Omitting Six Devices Talking to No More Than One Server). Green Area Highlights 120-hour Server Bootstrapping Period.

cannot do that here because these capture do not contain the device bootup traffic that we need to set T_{sp} values. (The 10-day measurement we use to select T_{sp} is different from the measurement we use to validate IoTSTEED later in §3.)

2.2.4 Second-Round Learning: Server Expansion. After server bootstrapping period, IoTSTEED only considers a server benign if its DNS domains resemble one of three classes of common IoT servers judged per-class rule below.

Manufacturer servers are servers run by IoT manufacturers to implement core IoT functions such as remote controlling and device monitoring. Manufacturer servers can usually be identified by their manufacturer-owned DNS domain. (IoTSTEED already knows at least a list of potential manufacturer names for each detected IoT devices in §2.1.) IoTSTEED considers a server name N as a manufacturer server for IoT device D if any of D's potential manufacturer name is a substring of N's domain (regardless of case). We define domain of a URL as the immediate left neighbor of the URL's public suffix. (We identify public suffix based on the list from Mozilla Foundation [56]).

Third-party servers are servers ran by non-manufacturers that provide services such as time (NTP) services and news services to IoT devices. We find it challenging to identify third-party servers because they could be specific to device types which we do not know. IoTSTEED thus only identifies two groups of third-party servers. The first group of servers are those providing NTP (time)

Tuya	Evrythng	PubNub	Xively	Azure IoT
tuyacn	evrythng	pubnub	xively	azure-devices
tuyaus		pndsn		
tuyaeu				

Table 2: Eight Domains from Five IoT Platforms

services, which we find common for all types of IoT devices to talk to. IoTSTEED identifies NTP servers by looking for servers with either well-known NTP domains (nist.gov and ntp.org) or string "time" or "ntp" in their sub-domain. The second group of servers are those run by the same organizations as some bootstrapped third-party servers. IoTSTEED identifies these servers by their use of bootstrapped third-party-server domains.

Platform servers are special third-party servers that allow manufacturers to implement core IoT functions without setting up their own servers. Platform servers can be identified by their platform-specific domain names. IoTSTEED currently looks for eight domains from five IoT platforms, as summarized in Table 2. Four of these IoT platforms (TuYa, Evrythng, PubNub and Xively) are reported by IoT inspector project [36] based on traffic from 44,956 IoT devices of 53 manufacturers. We also include the IoT platform from cloud provider Microsoft Azure since it has a unique domain ("azure-devices"). We do not include IoT platforms from other cloud providers (Amazon AWS and Google cloud) because their IoT platforms share the same domains ("amazonaws" and "googleapis") with their other cloud services and if IoTSTEED considers these domains benign, IoTSTEED risks allowing DDoS attacks to all their other clouds services.

Our second-round learning does not apply for IP-accessed servers due to their lack of DNS names. For these servers, if they fail first-round learning, IoTSTEED considers them malicious unless they are visited by devices whose IP-accessing filtering get turned off in §2.3.

To improve the applicability of second-round learning to IP-accessed servers, IoTSTEED keeps monitoring if any queried server names get resolved to any IP-accessed server IPs: if found, IoTSTEED assigns this server name to this IP-accessed server and re-learn this IP-accessed server with both round of learning as if it is a name-accessed server.

2.3 Traffic Filtering

IoTSTEED defends potential DDoS attack by passing traffic between IoT devices and IoT servers learned in §2.2.4 and dropping IoT traffic to and from all other servers. Upon dropping traffic from some IoT devices, IoTSTEED notifies device owners about the potential device compromise through its user interface.

For devices that show no bootstrapping behavior and instead keep talking to new IP-accessed servers, IoTSTEED turns off their IP-accessing filtering by passing all traffic between them and their IP-accessed servers. By doing so, IoTSTEED avoids dropping benign traffic to IP-accessed IoT servers that it fails to learn (false positives) but risk allowing these devices to attack server IPs (false negatives).

To detect devices that show no bootstrapping behavior, IoTSTEED looks for devices that keep talking to new IP-accessed servers both during and after server bootstrapping period (judged by Equation 1). We use $A(x)$ to annotate the number of distinct IP-accessed servers a given IoT device D talk to between period $[T_{bt}^D, T_{bt}^D + x)$, excluding whitelisted server IPs in §2.2.1. IoTSTEED considers D as lacking bootstrapping if at any time $T_{bt}^D + t$ after server bootstrapping (meaning $t > T_{sp}^{abi}$), the average rate that D talks to new IP-accessed servers after bootstrapping (R_t in Equation 1) is larger than a threshold. We set this threshold as a fraction (r in Equation 1, empirically set as 50%) of the average rate that D talks to new IP-accessed servers during server bootstrapping period (R_{sp} in Equation 1). IoTSTEED does not turn off IP-accessing filtering for devices that do not normally visit IP-accessed servers (judged by $A(T_{sp}^{abi}) \leq 2$ where 2 is empirical) because it is suspicious if they suddenly talks to some servers by IPs.

$$R_t > R_{sp} \times r \quad \text{where} \quad R_t = \frac{A(t) - A(T_{sp}^{abi})}{t - T_{sp}^{abi}} \quad \text{and} \quad R_{sp} = \frac{A(T_{sp}^{abi})}{T_{sp}^{abi}} \tag{1}$$

A compromised device D may try to evade IoTSTEED's defense by intentionally probing many server IPs and causing IoTSTEED to turn off its IP-accessing filtering. One possible way to mitigate this evasion is to configure IoTSTEED to turn IP-accessing filtering back on if D talk to new IP-accessed servers at too fast a rate after bootstrapping (for example, when $R_t > 10R_{sp}$).

2.4 Deployment Incentives

Both IoT owners and ISPs have reasons to deploy IoTSTEED.

IoTSTEED incentivizes IoT owners to run IoTSTEED in their home routers by protecting them from the potential privacy and security breach resulted from compromised IoT devices. For example, a compromised IP camera may leak live footage [67, 81], an hacked smart lock might lead to robbery [73] and a hacked smart oven could potentially cause house fire [7]. IoTSTEED protects IoT owners by constantly monitoring their IoT devices (§2.1 and §2.2) and notifying them about device compromises (§2.3). IoTSTEED also prevents IoT devices from talking to suspicious servers (§2.3) which mitigate the risks of IoT-related privacy breach (considering, for example, compromised IP cameras may talk to adversarial servers for transmitting video footage).

IoTSTEED incentivizes ISPs to pre-install IoTSTEED in their customer premises equipment (CPEs) from two aspects. First is value-added service: by pre-installing IoTSTEED (which protects their customers from compromised IoT devices), ISP is effectively providing an IoT security service. (Survey shows two-thirds of households with up to ten IoT devices are willing to pay an average of $6.90 per month for IoT security services [48].) Second is bandwidth saving: by rejecting IoT-based DDoS traffic at CPEs, ISPs save their bandwidth for legitimate user traffic.

2.5 Countermeasures by Knowledgeable Adversaries

DDoS attacks are launched by criminals who will seek to avoid detection. We next discuss four possible ways a knowledgeable adversary would try to evade IoTSTEED's defense. While they show

how an knowledgeable adversary can reduce IoTSTEED's effectiveness, these countermeasures either are difficult, have limited applicability, or weaken attacks.

First, a bot master could exploit first-round server learning (§2.2.3) by launching attacks during bots' server bootstrapping period and causing IoTSTEED to incorrectly learn attacks as valid behavior. Such evasion is unlikely in practice. Our bootstrapping period is a relatively short time window (2 hours or 120 hours after device bootup, §2.2.3). It is challenging for a bot master to infect IoT devices, rent out these devices to customers as part of DDoS-for-a-service infrastructure and launch attacks via these devices all within this time window.

Second, a bot master could exploit second-round server learning (§2.2.4) by launching attacks to the three types of common IoT servers IoTSTEED considers benign in §2.2.3 (such as servers run by attacking devices' manufacturers) and causing IoTSTEED to pass these attacks. IoTSTEED indeed cannot defend bots from attacking these common IoT servers. However by filtering attacks to all other servers, IoTSTEED still effectively breaks the economy of running DDoS as a service. The rationale is that to monetize DDoS infrastructure, a bot master needs to be able to attack any servers and not just a few common IoT servers.

Third, a bot master could also exploit traffic filtering (§2.3) by using bots whose IP-accessing filtering are off to attack IP-accessed servers and surpassing IoTSTEED's defense. While IoTSTEED works on most devices (11 or 12 of the 14 devices tested in §3 and §4), it does not work well on these a few devices. In addition, if UPnP is disabled, some of these devices become defendable (§4),.

Lastly, a bot master could exploit device detection (§2.1) by disguising bots as non-IoT devices and bypassing IoTSTEED's defense (since IoTSTEED does not filter non-IoT traffic). There are two ways to disguise bots as non-IoT devices: one could spoof bots' MAC addresses with some non-IoT MAC addresses (recalling §2.1.3); one could also make bots query more than T_{svr} server names and cause IoTSTEED to re-classify these bots as non-IoT devices (recalling §2.1.4). While disguising bots as non-IoT devices evades IoTSTEED's defense, IoTSTEED adds the burden of disguise to bot makers, making bot operation harder. The need to disguise bots also weakens the attacks by making bots more suspicious. Disguised bots could potentially be identified by the pool of non-IoT MAC addresses bot master use for MAC spoofing or the pool of server names that bot master makes bots query.

Potentially, we could make the last exploitation difficult to implement by requiring IoT owners to manually specify MAC addresses of their non-IoT devices when starting IoTSTEED and treating the rest MAC addresses as IoT devices. This way, to disguise bots as non-IoT device, a bot master needs to know MAC address of other non-IoT devices in the same LAN and spoof their bots with these specific non-IoT MAC addresses. (We do not currently do so to minimize manual operations required from IoT owners.)

3 VALIDATION BY TRACE REPLAY

We validate the correctness of IoTSTEED with replay of off-line traffic capture. We first validate IoTSTEED's accuracy and false positives (FPs) in detecting devices, learning server and filtering traffic with replay of 10 days of benign traffic captured from an IoT access network (§3.1). We then validate IoTSTEED's true positives (TPs) and false negatives (FNs) in filtering attack traffic with replay of real-world DDoS traffic (§3.2).

We use off-line traffic capture because they enable testing IoTSTEED with real-world DDoS traffic by replaying DDoS traffic capture (§3.2). In §4, we test IoTSTEED's accuracy, FPs, TPs and FNs in detecting devices, learning servers and filtering traffic with live traffic from an IoT access network.

3.1 False Positives with Benign Traffic

To understand IoTSTEED's accuracy in detecting devices (the fraction of IoT and non-IoT devices correctly detected) and FPs in learning servers (flagging of benign servers as suspicious) and filtering traffic (dropping of benign packets), we capture 10-day benign traffic from an IoT access network and run IoTSTEED with replay of these traffic. We show IoTSTEED correctly detect all 14 IoT and 7 non-IoT devices in this network (100% accuracy) and maintains low false-positive rate: flagging 2% of 642 IoT servers as suspicious and dropping 0.45% of about seven million benign IoT packets.

Experiment Setup: To test IoTSTEED with benign traffic, we set up an experimental IoT access network by placing 14 IoT devices (Table 1) and seven non-IoT devices (two mobile phones, two tablets and three laptops) in a wireless LAN behind a NAT router. (Our IoT devices, as in Table 1, are mostly IP cameras because IP cameras are used in large-scale DDoS attacks [55].) To simulate running IoTSTEED inside the NAT router, we capture traffic between the access network and the Internet by running tcpdump in NAT router (with UPnP service on) for 10 days. Since we require rebooting existing devices before starting IoTSTEED (§2.2.3), we shut off our IoT devices and boot them after tcpdump begins. We interact with our IoT devices daily from one of our mobile phones. (By testing with only our 14 IoT devices, our experiment is limited in device coverage. In principle, we could scale up by crowd-sourcing traffic capture from IoT devices owned by others, as shown in [36].)

Accuracy in Device Detection: We first test IoTSTEED's accuracy in detecting devices (§2.1). Our definition of accuracy is $(TP + TN)/(TP + TN + FP + FN)$ where we treat IoT devices as positives and non-IoT devices as negatives. To get IoTSTEED's accuracy, we compare devices it detects with ground truth and finding the fraction of IoT and non-IoT devices it correctly detects.

We show IoTSTEED correctly detects all 14 IoT devices and infers their manufacturers. IoTSTEED infers both Dahua and Amcrest as Amcrest_Cam's potential manufacturers because this device's MAC lookup result shows "Dahua" which is both an IoT manufacturer and a relative (OEM) to "Amcrest", recalling rules from §2.1.3.

We show IoTSTEED correctly identifies all seven non-IoT devices, resulting in overall 100% accuracy in device detection. IoTSTEED initially classifies six of these non-IoT devices as IoT because they come from IoT vendors (five from Apple and one from Samsung). IoTSTEED later re-classifies them as non-IoT since it observes that they query more than T_{svr} server names (§2.1.4). We show this initial mis-classification causes incorrect packet loss later this section.

False Positives in Server Learning: We next examine IoTSTEED's FPs in server learning (§2.2) and show it maintains low false-positive rate: flagging a small fraction (12 out of 642 or 2%) of IoT servers as malicious.

ground truth IoT servers from 14 IoT devices	642	(100%)	
whitelisted server IPs (all correctly identified)	8	(1%)	
enter first-round learning (all correctly identified)	464	(72%)	
enter second-round learning	170	(26%)	(100%)
correctly identified	158	(25%)	(93%)
detected by 3rd-party-svr rule	12	(2%)	(7%)
visited by devs without IP-acs fltr	146	(23%)	(86%)
mis-identified as malicious	12	(2%)	(7%)

Table 3: Server Learning Breakdown with Benign IoT Traffic

We breakdown server learning results by rounds in Table 3 to understand what cause the a few FPs. We show that the first-round learning causes no FP: it tests most (464 or 72%) of 642 ground truth IoT server and correctly identifies all of them (Table 3). Our second-round learning causes all the FPs by mis-identifying a small fraction (12 or 7%) of the 170 IoT servers it tests as malicious. Eight of these 12 FPs are IP-accessed servers and the rest four are name-accessed servers whose domains ("google" and "opendns") do not resemble common IoT servers, judged by rules in §2.2.4.

Lastly, we show that turning off IP-accessing filtering is crucial for maintaining IoTSTEED's low FPs in learning servers. IoTSTEED turns off three devices' IP-accessing filtering (Samsung_Cam, D-Link_Cam and Foscam_Cam) because they constantly talk to new IP-accessed servers (§2.3). We find that most (146 or 86%) of the 170 IoT servers tested by second-round learning are classified benign (TPs) only because they are visited by these three devices directly by IP (Table 3). If IoTSTEED keeps these three device's IP-accessing filtering on, these 146 IP-accessed servers would be flagged as malicious (FPs, recalling §2.2.4), resulting in a high false-positive rate of 25% (158 out of 642) in learning servers. (In §3.2, we show that the tradeoff of turning off IP-accessing filtering is that IoTSTEED risks allowing attacks to IP-accessed servers.)

False Positives in Traffic Filtering: We next examine IoTSTEED's FPs in filtering traffic (§2.3). We show IoTSTEED's false-positive rate is low, dropping only a tiny fraction (33,183 or 0.45%) of about seven million packets our IoT devices send and receive in this 10-day measurement.

We show that out of these 33,183 false-positive packet losses, most (90.72% or 30,104) are because IoTSTEED misclassifies 12 IoT servers as malicious (Table 3). We show that the rest false-positive packet losses (7.83%, 2,597 out of 33,183) are because IoTSTEED initially mis-classifies six non-IoT devices as IoT and filters these non-IoT devices' traffic as if they are IoT traffic. After IoTSTEED later corrects mis-classification of these six non-IoT devices, it no longer filters their traffic (recalling IoTSTEED ignores non-IoT traffic). In §4, we show that we could avoid these packet losses due to initial mis-classification by listing all non-IoT devices' MAC addresses as exceptions (whose packets IoTSTEED will ignore) when starting IoTSTEED.

3.2 True Positives and False Negatives with Attack Traffic

To understand IoTSTEED's TPs and FNs in filtering attack traffic, we spoof traffic capture of real-world DDoS attacks and run IoTSTEED with replay of these spoofed DDoS traffic.

Victims	Access Type	Simulated Attackers	Traffic Type	Start After Bootstrap?	Filtering Decision
B root	IP	Amcrest_Cam	TCP	Yes	Drop
B root	IP	Amcrest_Cam	DNS	Yes	Drop
B root	IP	Amcrest_Cam	TCP	No	*Pass*
B root	IP	Foscam_Cam	TCP	Yes	*Pass*
B root	IP	a non-IoT dev	TCP	Yes	*Pass*
Krebs	Name	Amcrest_Cam	TCP	Yes	Drop
Krebs	Name	Philips_Bulb	TCP	Yes	Drop
Philips	Name	Philips_Bulb	TCP	Yes	*Pass*
Krebs	Name	HP_Printer	TCP	Yes	Drop
Krebs	Name	Dyson_Purifier	TCP	Yes	Drop

Table 4: Simulated Attacks in Trace-replay Validation

Replay of Spoofed Real-world DDoS Traffic: We test IoTSTEED with attack traffic from two real-world DDoS events captured at B-root DNS server (simply "B root" hereafter): a DNS query flooding event in December 2015 and a TCP SYN flooding event in June 2016. We first simulate ten IoT-based DDoS attacks (each a row in Table 4) by spoofing attack traffic capture from these two DDoS events so that the attacks appear coming from our IoT devices. (Among these ten simulated attacks, four are examples of the four potential countermeasures to IoTSTEED discussed in §2.5.) We then test IoTSTEED with replay of each of these simulated IoT-based DDoS attacks together with 10-day benign traffic capture from §3.1.

We first simulate five IoT-based DDoS attacks to an IP-accessed server: B root. We simulate attacks to B root because IoT-based DDoS attacks have frequently targeted DNS servers [17]. We do not simulate attacks to other IP-accessed servers because IoTSTEED only examines when an IP-accessed server gets accessed (§2.2.3) and by whom (§2.2.4) rather than its exact server IP. To simulate a certain IoT device D attacking B root at time t via TCP SYN flooding (simulating DNS-query flooding from D is similar), we first extract one random attacker's DDoS packets from 15-minute sample of TCP SYN flooding event (referred as TCP "attacker capture"). We then replace the source MAC and IP addresses in TCP attacker capture with MAC and LAN IP of D and shifting timestamp of all DDoS packets in TCP attacker capture to right after t (called "spoofed attacker capture"). Since the two DDoS events we use are captured at victim (B root) and do not include potential DNS queries from attackers about victim, our spoofed attacker capture simulates device D attacking B root directly by IP. We summarize the five IoT-based DDoS attacks to B root we simulate in top half of Table 4. Four of them are based on spoofing TCP attacker capture with four combinations of attackers and attacking time: Amcrest_Cam, Foscam_Cam and an non-IoT device attacking after server bootstrapping and Amcrest_Cam attacking during bootstrapping. (We test IoTSTEED with an non-IoT attack to simulate defending IoT devices disguised as non-IoT, one countermeasure to IoTSTEED from §2.5.) The remaining one simulated attack is based on spoofing DNS attacker capture with one attacking device (Amcrest_Cam) and time (after bootstrapping).

We next simulate five attacks to two name-accessed servers: www.krebsonsecurity.com and www.philips.com (shortened as "Krebs" and "Philips" hereafter.) We simulate attacks to Philips, a manufacturer server for Philips_Bulb, to test IoTSTEED's defense of attacks

to common IoT servers (recalling §2.2.4). We simulate attacks to Krebs, victim of an IoT-based DDoS attack in 2016 [44], to test IoTSTEED's defense of attacks to all other name-accessed servers. To simulate a given IoT device D attacking Krebs at time t via TCP SYN flooding (simulating attacks to Philips is similar), We first generate a spoofed TCP attacker capture for D (which simulate D TCP SYN flooding B root at time t) as we discussed above. We then replace victim IP in spoofed TCP attacker capture (B-root IP) with Krebs' IP. Lastly, we inject forged DNS traffic (containing type-A DNS query from D about Krebs and corresponding DNS replies) to the beginning of this spoofed TCP attacker capture. We summarize the five IoT-based DDoS attacks to Krebs and Philips we simulate in the bottom half of Table 4. Our five simulated attacks are based on spoofing TCP attacker capture with five different combinations of attackers and victims: Amcrest_Cam, Philips_Bulb, HP_Printer and Dyson_Purifier attacking Krebs and Philips_Bulb attacking Philips.

True Positives with attack traffic: We show IoTSTEED defends all attacks except the four based on countermeasures from §2.5 (see the six attacks with "Drop" filtering decision in Table 4). We show IoTSTEED defends these six attacks regardless of their attacking devices (four different devices Table 4), traffic types (TCP SYN flooding and DNS query flooding), victims (B root and Krebs) and accessing types for victims (IP-accessed and name-accessed). In router deployment §4, we extend this observation by showing that IoTSTEED defends attacks regardless of their packet rates.

False Negatives with Attack Traffic: We confirm IoTSTEED indeed cannot mitigates the four types of attacks discussed in §2.5, contributing to the four FNs we observe in Table 4 (see the four "passed" attacks). We also confirm our conclusion from §2.5 that these countermeasures either are difficult, have limited applicability, or weaken attacks.

First, we show that IoTSTEED cannot defend attacks launched during server bootstrapping period of the attacking devices, which contribute to the passing of attacks from Amcrest_Cam to B root during bootstrapping (third B-root attack in Table 4). However this type of attacks are not likely to happen in practice since bootstrapping period is relatively short (see §2.5 for details.)

Second, we show IoTSTEED cannot defend devices from attacking the three class of common IoT servers it considers benign in §2.2.4, which causes the missing of attacks from Philips_Bulb to Philips (see the attack to Philips in Table 4). While IoTSTEED cannot defend attacks to a few common IoT servers, IoTSTEED still defend the rest majority of servers and break economy of running commercial DDoS attacks (as shown in §2.5).

Third, we show that IoTSTEED cannot defend devices without IP-accessing filtering from attacking IP-accessed server, which contribute to the passing of attacks from Foscam_Cam whose IP-accessing filtering is off (forth B-root attack in Table 4). We argue that these devices are in minority (3 of 14 IoT devices in §3) and that some of them (including Foscam_Cam, as shown later in §4) would become defendable once disabling UPnP service in router (recalling §2.5).

Lastly, we show IoTSTEED cannot defend attacks from IoT devices disguised as non-IoT devices, which contributes to the passing of attacks from an non-IoT device to B root (last B-root attack in Table 4). While disguising IoT devices does evade IoTSTEED's defense, it weaken the resulting IoT-based DDoS attack by making the attack harder to implement and making the bots easier to identify, as detailed in §2.5.

Victims	Access Type	Simulated Attackers	Traffic Type	Start After Bootstrap?	Filtering Decision
Univ Svrs	IP	Amcrest_Cam	Slow TCP	No	*Pass*
Univ Svrs	IP	Foscam_Cam	Slow TCP	Yes	Drop
Univ Svrs	IP	Philips_Bulb	Slow TCP	Yes	Drop
Univ Svrs	IP	TPLink_Plug	Fast TCP	Yes	Drop

Table 5: Simulated Attacks in Router-deployment Validation

In summary, our experiment results suggest IoTSTEED could mitigate all except the four types of attacks discussed in §2.5 regardless of the attacks' traffic types, attacking devices and victims.

4 VALIDATION BY ROUTER DEPLOYMENT

Having tested IoTSTEED with replay of off-line traffic capture (§3), we next deploy it on-line in the NAT router of an IoT access network for 10 days. We show IoTSTEED works in router deployment similar to trace-replay validation: with reasonable run-time overhead, few false positives (FPs) with benign traffic and similar true positives (TPs) and false negatives (FNs) with attack traffic.

Experiment Setup: We deploy IoTSTEED in the NAT router of the experimental IoT access network from §3. (The NAT router is a Linksys WRT1900ACS router running OpenWRT version 19.07.1 [59].) We add one extra Linux personal computer (PC) to this network to simulate IoT-based DDoS attacks. Similar to trace-replay validation (§3), our deployment experiment lasts 10 days. We shut off all IoT devices initially and boot them up after IoTSTEED starts. We interact with IoT devices daily with the same mobile phone as in §3.

We simulate four IoT-based DDoS attacks during 10-day deployment, as summarized in Table 5. We run hping3 [34] from one Linux laptop in this IoT access network (with one "fast" TCP SYN flooding attack of 1000 packets/s and three "slow" TCP SYN attacks of 10 packets/s) and spoof this laptop's MAC and IP addresses with those of certain IoT devices (the attackers in Table 5). We block all inbound and outbound traffic to this laptop except traffic to DDoS victim to prevent laptop from talking to non-victim PC-oriented servers using spoofed IoT MAC addresses and confusing IoTSTEED into thinking that some IoT devices are talking to these PC-oriented servers. (We cannot simulate attacks to name-accessed servers because if we allow DNS traffic from this Linux laptop, it could DNS query non-victim PC-oriented servers, such as connectivity-check.ubuntu.com, with spoofed IoT MAC address and confuse IoTSTEED.)

Lastly, we apply two tweaks to our router-deployment experiment to validate our claims earlier in trace-replay validation that they could reduce FPs (§3.1) and FNs (§3.2) in filtering traffic. First, we list all non-IoT devices' MAC addresses as exceptions (whose packets IoTSTEED would ignore) when starting IoTSTEED. Our goal is to validate the claim that doing so could reduce FPs in traffic filtering (caused by initial mis-classification of some non-IoT devices as IoT, recalling §3.1). Second, we disable UPnP service in NAT router during our 10-day deployment. Disabling UPnP allows us

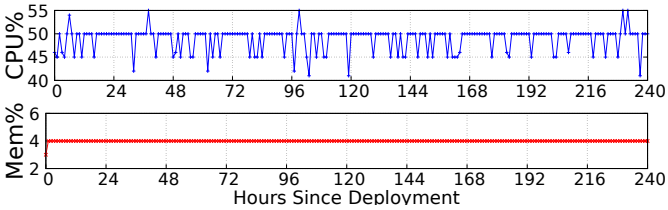

Figure 5: IoTSTEED's Per-hour CPU and Memory Usage during Router-deployment Validation

to validate our claim that without UPnP, devices like Foscam_Cam will stop constantly visiting new IP-accessed servers and causing potential FNs in filtering DDoS attacks (§3.2).

Measuring Run-time Overhead: We measure memory and CPU usage of IoTSTEED every hour during this 10-day deployment. We show IoTSTEED uses a small amount (in average 4%) of the 512MB memory and about half (in average 49%) of the 1.6 GHz dual-core CPU in this NAT router and its memory and CPU usages are quite stable, as illustrated in Figure 5.

We confirm that IoTSTEED's CPU usage does not slow down router's packet forwarding and thus even though half the CPU is busy, the router's user-visible performance is unaffected. We run Internet speed test (www.speedtest.net) from one laptop in this access network and confirm that this laptop's peak download and upload speeds with IoTSTEED running (113 Mb/s and 11 Mb/s, averaged over 10 tests) is roughly identical to those without IoTSTEED running (114 Mb/s and 11 Mb/s, averaged over 10 tests). (As CPUs in home routers keep growing faster, we expect IoTSTEED's CPU usage to decrease over time.)

False Positives with Benign Traffic: We confirm IoTSTEED's accuracy in detecting devices and FPs in learning servers and filtering traffic during router deployment are similar to what we report in trace-replay validation (§3.1). IoTSTEED correctly detects all 14 IoT devices and infers their manufacturers (100% accuracy). It maintains low false-positive rate in learning servers: flagging a small amount (6 or 4%) of 139 IoT servers as suspicious. (Six FPs include five IP-accessed servers entering second-round learning and one name-accessed servers with domain "Google" that does not resemble common IoT servers.) It also shows low false-positive rate in filtering traffic: dropping a tiny fraction (8,769 or 0.07%) of 12 million benign IoT packets observed from this LAN.

We show all FPs in filtering traffic are because IoTSTEED flags six IoT servers as suspicious during server learning. Comparing to §3.1, we see no false-positive filtering caused by initial misclassification of non-IoT devices as IoT because we exclude non-IoT MAC addresses. We conclude that excluding non-IoT MAC addresses when starting IoTSTEED could reduce FPs in filtering traffic.

True Positives and False Negatives with Attack Traffic: We next examine IoTSTEED's TPs and FNs in filtering attack traffic during router deployment and show they are similar to what we observe in trace-replay validation. We confirm that IoTSTEED cannot defend attack during bootstrapping, which causes the one FN in Table 5. We confirm IoTSTEED mitigates all other three attacks tested regardless of the attacking devices (three different devices Table 5) and packet rates (1000 packets/s and 10 packets/s) of attacks.

We note that IoTSTEED successfully mitigates the attack from Foscam_Cam during router deployment while passes a similar attack from Foscam_Cam during trace-replay validation (§3.2) . We believe the reason for this difference is that we turn off UPnP service in deployment. In trace-replay validation, Foscam_Cam keeps responding to unsolicited probes from Internet scanners (a side effect of UPnP) and appears to be constantly visiting new IP-accessed servers, forcing IoTSTEED to turn off its IP-accessing filtering. In deployment, these Internet scanners cannot reach Foscam_Cam because without UPnP service, Foscam_Cam cannot set up static port mapping in router. As a result, IoTSTEED keeps IP-accessing filtering for Foscam_Cam on and mitigate its attack. We conclude that for some devices (such as Foscam_Cam, Samsung_Cam and D-Link_Cam in router deployment), we can prevent them from constantly contacting new IP-accessed servers and failing IoTSTEED's defense by disabling UPnP service in router.

Lastly, we show that even without UPnP service, a few devices could still keep visiting new IP-accessed servers and IoTSTEED cannot defend them from attacking IP-accessed servers. During router deployment, we find Belkin_Plug and Tenvis_Cam both talk to a few new IP-accessed servers after bootstrapping and cause IoTSTEED to turn off their IP-accessing filtering. For Belkin_Plug, we find most (12 or 80%) of its 15 IP-accessed servers to be STUN servers for NAT traversal, similar to what we observe in prior measurements (§2.2.3 and in §3). However we are not certain about why Belkin_Plug keeps talking to new STUN servers throughout router deployment but stops visiting new STUN servers in the middle of measurement in §3. We are also unclear about why Tenvis_Cam talks to a few new IP-accessed servers (3, not counting whitelisted server IPs in §2.2.1) after bootstrapping in router deployment but not in our prior measurements (§2.2.3 and §3).

5 RELATED WORK

Prior groups have studied detecting IoT devices and defending both IoT-based and traditional DDoS attacks (launched from PCs and servers).

5.1 IoT Device Detection

Several prior projects detect IoT devices with public IPs by active scanning [9, 16, 53, 69, 74]. In comparison, our detection uses passive measurement and thus could detect devices behind NATs (when running from NAT box). Other prior work also measures IoT devices passively and thus covers devices behind NAT [3, 29, 30, 33, 71]. However they either rely on pre-training with traffic from target devices [29, 30, 71] (our prior studies on general IoT detection [29, 30] fall into this category), or only cover IoT devices infected by certain malware [3, 33]. In comparison, our detection is based on MAC addresses and requires no such pre-training with target devices' traffic. Our detection also applies to general IoT devices instead of just the compromised ones.

5.2 IoT-Based DDoS Defense

Traffic Endpoint as Signals: Similar to our work, prior work also explore the idea of detecting IoT-based DDoS traffic based on traffic endpoints. They either operate near the IoT devices or near the remote endpoints IoT devices talk to.

Several groups propose detecting IoT-based DDoS traffic based on traffic endpoints from the edge routers of IoT access networks [14, 32, 61]. The preliminary measurement study from Brown University shows it is feasible to detect DDoS traffic from IoT devices by whitelisting the benign endpoints they talk to, without providing a method to build this whitelist [14]. Along the same line, work from University of New South Wales proposes building this whitelist of benign endpoints based on manufacturer usage description (MUD, an IETF draft [45]) profiles provided by MUD-compliant IoT devices [32]. Different from both work, we provide concrete mechanisms to build whitelists of benign endpoints that IoT devices talk to (recalling IoT servers in §2.2), without relying on the currently non-existent MUD-compliant IoT devices. Bogazici University detects compromised IoT devices by identifying devices that send TCP SYN packets to many different endpoints in a short interval without receiving as much positive responses [61]. Unlike their focus on TCP SYN flooding traffic, we detect all types of DDoS traffic not sent to benign endpoints.

One group detects malicious IoT traffic by measuring from the remote endpoints IoT devices talk to [74]. Work from Concordia University ([74]) infers compromised IoT devices in the public Internet by identifying the fraction of IoT devices detected by Shodan ([69]) that send packets to allocated but un-used IPs monitored by CAIDA (as known as darknet [6]). In comparison, our work have very different coverage: we cover compromised IoT devices in the access network we monitor (potentially with private IP addresses) while they cover a subset of compromised IoT devices on public Internet. By whitelisting benign endpoints, we cover malicious IoT traffic to all suspicious endpoints, unlike their focus on malicious IoT traffic to a specific group of suspicious endpoints: the CAIDA darknet IPs.

Traffic Flow Statistics as Signals: Several prior work detect IoT-based DDoS traffic based on traffic flow statistics (such as packet rates and packet sizes) from the edge router of an IoT access network (such as the NAT router of a LAN) [15, 51, 58]. They detect IoT-based DDoS traffic using machine learning (ML) models trained with either a mixture of benign IoT traffic and simulated attack traffic (binary classification that detects DDoS by looking for similarity to known attack traffic [15]) or with only benign IoT traffic (anomaly detection that detects DDoS by looking for deviations from known benign traffic [51, 58]). Unlike their signals based on traffic flow statistics, we use traffic endpoints' first-visit time and identities as signal. Comparing to their detection techniques of ML-based binary classification [15] and anomaly detection [51, 58], we use a different technique: heuristic-based rules. While they all assume an IoT-only access network, IoTSTEED could separate IoT from non-IoT devices and could operate in realistic access networks with both IoT and non-IoT devices. IoTSTEED also covers DDoS traffic of different types (such as TCP SYN flooding and DNS query flooding) and flow statistics (such as packet rates) by dropping all traffic not sent to benign endpoints. In comparison, ML-based binary classification only detects attacks similar to known attack traffic seen in model training [15]. Although in principle, ML-based anomaly detection could identify malicious traffic of different types and flow statistics by looking for deviations from known benign traffic [51, 58], prior work have shown that ML models are not good at detecting such deviations especially when dealing with highly-variable real-world network traffic [26, 72]. Lastly, their methods are computationally heavy and are not likely to run from resource-constraint home router. In comparison, IoTSTEED is light-weight and could run in commodity home router with reasonable overhead (§4).

Other Signals: Other prior work detect compromised IoT devices and defend malicious IoT-based DDoS traffic with other signals [13, 38]. Work from IFFAR detects compromised IoT sensors reporting altered measurements by finding outliers in measurements reported by a pool of homogeneous IoT sensors [13]. In comparison, IoTSTEED applies to all types of IoT access network instead of just networks of homogeneous IoT sensors. Work from National University of Singapore [38] mitigates IoT-based DDoS attacks to a given server by setting static traffic quotas in this server for each contacting IoT device and dropping excessive packets. In comparison, IoTSTEED do not require access to victim servers.

The industry is also working on detecting malicious IoT traffic. Telekom Security of T-Systems detects compromised IoT devices by monitoring darknet, their worldwide honeypot networks and their CPEs [23]. Upon detection, they filter attack traffic at ISP boarder and CPEs and inform customers about their compromised IoT devices. In comparison, IoTSTEED is local to LAN and do not requires monitoring darknet and accessing global honeypot networks.

5.3 Traditional DDoS Defense

Prior to IoT-based DDoS attacks, DDoS attacks launched from traditional network devices such as PC and servers has been studied widely.

Defense at Bot Side: similar to our work, prior work propose defending traditional DDoS traffic at bot side [18, 24, 54]. D-WARD detects DDoS attacks at bot side by comparing traffic flow statistics from and to the access network of bots with pre-defined models for normal flow statistics [54]. FireCol puts multiple intrusion prevention systems (IPS) close to the access network of bots and detects DDoS based on traffic bandwidths measured from these IPSes [18]. MULTOPS detects DDoS attacks overloading network bandwidth of victims by identifying disproportional difference between packet rate coming from and going back to the bots' access network [24]. Different from their focus on traffic flow statistics such as packet rates and ratio of number of packets sent and received, our work focuses on traffic endpoints such as their first-visit time and DNS names.

Defense at Intermediate Network: Prior work also studies defending DDoS attack at the intermediate network between bots and victims. Work from AT&T Labs detects DDoS traffic from intermediate network by identifying aggregates of network flows that cause network link congestion. Their detection relies on traffic flow statistics such as packet arrival rates and packet dropping history [50]. In comparison, IoTSTEED detects DDoS traffic with traffic endpoints as signals and operates at bot side.

Defense at Victim Side: Other prior work studies defending DDoS traffic at victim network.

Prior work defends network and transport-layer DDoS attacks that consumes bandwidth and other resources of victims (such as ICMP flooding and TCP SYN flooding) with techniques such as statistical anomaly detection (which detects DDoS by identifying anomaly in certain traffic flow statistics) [75], flow imbalance

heuristic (which looks for unbalanced packet rates between the incoming and outgoing traffic flows) [1] and TCP SYN cookies [4].

Prior work defends application-layer DDoS attacks targeting web applications (such as HTTP GET flooding) with techniques such as Turing test (which distinguishes legitimate human users from machines) [41, 63], moving-target techniques (which moves target web application around a pool of servers to increase uncertainty for the attacker) [39, 43, 78], domain-helps-domain collaboration (which allows a domain to direct excessive traffic to other trusted external domains for DDoS filtering) [65] and also statistical anomaly detection [57, 64].

Prior work also defends DDoS attacks targeting software-defined network (SDN) by deploying new packet scheduling algorithms in SDN controllers [35, 46], running traffic-statistic-based DDoS detection algorithms in SDN [40, 62, 76] and improving SDN architecture [8, 80].

Different from these prior work that focus on defending DDoS attacks at victim side, IoTSTEED defends DDoS traffic at bot side.

6 CONCLUSION

We propose IoTSTEED, a system that defends IoT-based DDoS attacks at bot-side. IoTSTEED detects IoT devices in its deployed network, learns their benign servers and filters their traffic to other servers as potential DDoS attacks. We validate IoTSTEED with replay of 10-day benign traffic captured from an IoT access network and simulated IoT-based DDoS attacks. We show IoTSTEED could correctly detect the 14 LAN IoT and 6 non-IoT devices in this access network (100% accuracy). We show IoTSTEED maintains low false-positive rate in learning servers (2%) and filtering traffic (0.45%). Experiment results also show IoTSTEED mitigates all typical attacks, regardless of the attacks' traffic types, attacking devices and victims; an intelligent adversary can design to avoid detection in a few cases, but at the cost of a weaker attack. Lastly, we deploy IoTSTEED in NAT router of an IoT access network for 10 days, showing reasonable resource usage (4% of 512MB memory) and verifying our testbed experiments for accuracy and learning in practice.

ACKNOWLEDGMENTS

Hang Guo and John Heidemann's work is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-17-2-0280. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

REFERENCES

[1] Abdelsayed, S., Glimsholt, D., Leckie, C., Ryan, S., and Shami, S. An efficient filter for denial-of-service bandwidth attacks. In *IEEE Global Telecommunications Conference* (Dec 2003), vol. 3, pp. 1353–1357 vol.3.

[2] Alrawi, O., Lever, C., Antonakakis, M., and Monrose, F. SoK: Security evaluation of home-based IoT deployments. In *2019 IEEE Symposium on Security and Privacy (SP)* (May 2019), pp. 1362–1380.

[3] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., and Zhou, Y. Understanding the Mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)* (Vancouver, BC, 2017), USENIX Association, pp. 1093–1110.

[4] Bernstein, D. J. TCP SYN cookies. http://cr.yp.to/syncookies.html.

[5] Bray, H. Akamai breaks ties with security expert. https://www.bostonglobe.com/business/2016/09/23/cybercrooks-akamai/qOAhvHoohJcmkxIwg5ChKO/story.html.

[6] CAIDA. The UCSD network telescope. https://www.caida.org/projects/network_telescope/.

[7] Carman, A. Smart ovens have been turning on overnight and preheating to 400 degrees. https://www.theverge.com/2019/8/14/20802774/june-smart-oven-remote-preheat-update-user-error.

[8] Chourishi, D., Miri, A., Milić, M., and Ismaeel, S. Role-based multiple controllers for load balancing and security in SDN. In *IEEE Canada International Humanitarian Technology Conference (IHTC)* (May 2015), pp. 1–4.

[9] Chung, T., Liu, Y., Choffnes, D., Levin, D., Maggs, B. M., Mislove, A., and Wilson, C. Measuring and applying invalid SSL certificates: The silent majority. In *Proceedings of the 2016 Internet Measurement Conference* (New York, NY, USA, 2016), IMC '16, ACM, pp. 527–541.

[10] Constantin, L. Hackers found 47 new vulnerabilities in 23 IoT devices at DEF CON. https://www.csoonline.com/article/3119765/security/hackers-found-47-new-vulnerabilities-in-23-iot-devices-at-def-con.html.

[11] Costin, A., Zarras, A., and Francillon, A. Automated dynamic firmware analysis at scale: A case study on embedded web interfaces. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (New York, NY, USA, 2016), ASIA CCS '16, ACM, pp. 437–448.

[12] Day, M., Turner, G., and Drozdiak, N. Amazon workers are listening to what you tell alexa. https://www.bloomberg.com/news/articles/2019-04-10/is-anyone-listening-to-you-on-alexa-a-global-team-reviews-audio.

[13] de Souza, P. S. S., dos Santos Marques, W., Rossi, F. D., da Cunha Rodrigues, G., and Calheiros, R. N. Performance and accuracy trade-off analysis of techniques for anomaly detection in IoT sensors. In *2017 International Conference on Information Networking (ICOIN)* (Jan 2017), pp. 486–491.

[14] DeMarinis, N., and Fonseca, R. Toward usable network traffic policies for IoT devices in consumer networks. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy* (New York, NY, USA, 2017), IoTS&P '17, ACM, pp. 43–48.

[15] Doshi, R., Apthorpe, N., and Feamster, N. Machine learning DDoS detection for consumer Internet of Things devices. *CoRR abs/1804.04159* (2018).

[16] Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., and Halderman, J. A. A search engine backed by Internet-wide scanning. In *Proceedings of the ACM Conference on Computer and Communications Security* (Denver, CO, USA, Oct. 2015), ACM, pp. 542–553.

[17] Dyn. Dyn analysis summary of Friday October 21 attack. http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[18] Francois, J., Aib, I., and Boutaba, R. FireCol: A collaborative protection network for the detection of flooding DDoS attacks. *IEEE/ACM Transactions on Networking 20*, 6 (Dec 2012), 1828–1841.

[19] Gartner. Gartner says 5.8 billion enterprise and automotive IoT endpoints will be in use in 2020. https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io.

[20] Geenens, P. Assessing the threat the Reaper botnet poses to the Internet—what we know now. https://arstechnica.com/information-technology/2017/10/assessing-the-threat-the-reaper-botnet-poses-to-the-internet-what-we-know-now/.

[21] Geenens, P. Hajime: Analysis of a decentralized Internet worm for IoT devices. https://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf.

[22] Geenens, P. Hajime – sophisticated, flexible, thoughtfully designed and future-proof. https://blog.radware.com/security/2017/04/hajime-futureproof-botnet/.

[23] Giese, C. Detect & respond to IoT botnets as an ISP. https://www.first.org/resources/papers/conf2018/Giese-Christoph_FIRST_20180620.pdf.

[24] Gil, T. M., and Poletto, M. MULTOPS: A data-structure for bandwidth attack detection. In *Proceedings of the 10th Conference on USENIX Security Symposium* (Berkeley, CA, USA, 2001), SSYM, USENIX Association.

[25] GREENBERG, A. Hackers found a not-so-easy way to make the Amazon Echo a spy bug. https://www.wired.com/story/hackers-turn-amazon-echo-into-spy-bug/.

[26] Guo, H., Fan, X., Cao, A., Outhred, G., and Heidemann, J. Peek inside the closed world: Evaluating autoencoder-based detection of DDoS to cloud, 2019.

[27] Guo, H., and Heidemann, J. IoT_Operation_Traces-20200127. https://ant.isi.edu/datasets/iot/.

[28] Guo, H., and Heidemann, J. IoTSTEED source code. https://ant.isi.edu/software/iotsteed/index.html.

[29] Guo, H., and Heidemann, J. Detecting IoT devices in the Internet (extended). Tech. Rep. ISI-TR-726, USC/Information Sciences Institute, July 2018.

[30] Guo, H., and Heidemann, J. IP-based IoT device detection. In *Proceedings of the 2nd Workshop on IoT Security and Privacy* (aug 2018).

[31] Hamza, A., Gharakheili, H. H., Benson, T. A., and Sivaraman, V. Detecting volumetric attacks on IoT devices via SDN-based monitoring of MUD activity. In *Proceedings of the 2019 ACM Symposium on SDN Research* (New York, NY, USA, 2019), SOSR '19, Association for Computing Machinery, p. 36–48.

[32] Hamza, A., Gharakheili, H. H., and Sivaraman, V. Combining MUD policies

with SDN for IoT intrusion detection. In *Proceedings of the 2018 Workshop on IoT Security and Privacy* (New York, NY, USA, 2018), IoT S&P '18, ACM, pp. 1–7.

[33] Herwig, S., Harvey, K., Hughey, G., Roberts, R., and Levin, D. Measurement and analysis of Hajime, a peer-to-peer IoT botnet. In *Network and Distributed System Security Symposium (NDSS)* (Feb 2019).

[34] hping. Hping project. http://www.hping.org/hping3.html.

[35] Hsu, S., Chen, T., Chang, Y., Chen, S., Chao, H., Lin, T., and Shih, W. Design a hash-based control mechanism in vSwitch for software-defined networking environment. In *IEEE International Conference on Cluster Computing* (Sep. 2015), pp. 498–499.

[36] Huang, D. Y., Apthorpe, N., Acar, G., Li, F., and Feamster, N. IoT inspector: Crowdsourcing labeled network traffic from smart home devices at scale, 2019.

[37] IPVM. Dahua OEM directory. https://ipvm.com/reports/dahua-oem.

[38] Javaid, U., Siang, A. K., Aman, M. N., and Sikdar, B. Mitigating IoT device based DDoS attacks using blockchain. In *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems* (New York, NY, USA, 2018), CryBlock'18, ACM, pp. 71–76.

[39] Jia, Q., Sun, K., and Stavrou, A. MOTAG: Moving target defense against Internet denial of service attacks. In *22nd International Conference on Computer Communication and Networks (ICCCN)* (July 2013), pp. 1–9.

[40] Kalkan, K., Gür, G., and Alagöz, F. SDNScore: a statistical defense mechanism against DDoS attacks in SDN environment. In *IEEE Symposium on Computers and Communications (ISCC)* (July 2017), pp. 669–675.

[41] Kandula, S., Katabi, D., Jacob, M., and Berger, A. Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2* (Berkeley, CA, USA, 2005), NSDI'05, USENIX Association, pp. 287–300.

[42] Kelly, R. L. IoT Cybersecurity Improvement Act of 2019. https://www.congress.gov/bill/116th-congress/house-bill/1668/text.

[43] Khattab, S. M., Sangpachatanaruk, C., Melhem, R., ı Mosse, D., and Znati, T. Proactive server roaming for mitigating denial-of-service attacks. In *International Conference on Information Technology: Research and Education. ITRE* (Aug 2003), pp. 286–290.

[44] Krebs, B. KrebsOnSecurity hit with record DDoS. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/.

[45] Lear, E., Droms, R., and Romascanu, D. Manufacturer usage description specification. https://tools.ietf.org/html/rfc8520.

[46] Lim, S., Yang, S., Kim, Y., Yang, S., and Kim, H. Controller scheduling for continued SDN operation under DDoS attacks. *Electronics Letters 51*, 16 (2015), 1259–1261.

[47] Loshin, P. Details emerging on Dyn DNS DDoS attack, Mirai IoT botnet. blog http://searchsecurity.techtarget.com/news/450401962/Details-emerging-on-Dyn-DNS-DDoS-attack-Mirai-IoT-botnet, Oct. 2016.

[48] Ltd, A. New research shows that consumers are willing to pay for IoT services. https://www.allot.com/blog/new-research-shows-that-consumers-are-willing-to-pay-for-iot-services/.

[49] macaddress.io. MAC address vendor lookup library. https://macaddress.io/.

[50] Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., and Shenker, S. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev. 32*, 3 (July 2002), 62–73.

[51] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., and Elovici, Y. N-BaIoT—network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing 17*, 3 (Jul 2018), 12–22.

[52] Micro, T. Hide N Seek botnet uses Peer-to-Peer infrastructure to compromise IoT devices. https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/-hide-n-seek-botnet-uses-peer-to-peer-infrastructure-to-compromise-iot-devices.

[53] Mirian, A., Ma, Z., Adrian, D., Tischer, M., Chuenchujit, T., Yardley, T., Berthier, R., Mason, J., Durumeric, Z., Halderman, J. A., and Bailey, M. An Internet-wide view of ICS devices. In *Annual Conference on Privacy, Security and Trust (PST)* (Dec 2016).

[54] Mirkovic, J., Prier, G., and Reiher, P. Attacking DDoS at the source. In *10th IEEE International Conference on Network Protocols* (Nov 2002), pp. 312–321.

[55] Motherboard. 15 million hijacked cameras make an unprecedented botnet. https://motherboard.vice.com/en_us/article/8q8dab/15-million-connected-cameras-ddos-botnet-brian-krebs.

[56] Mozilla. Public suffix list from Mozilla foundation. https://www.publicsuffix.org/.

[57] Najafabadi, M. M., Khoshgoftaar, T. M., Calvert, C., and Kemp, C. User behavior anomaly detection for application layer DDoS attacks. In *IEEE International Conference on Information Reuse and Integration (IRI)* (Aug 2017), pp. 154–161.

[58] Nguyen, T. D., Marchal, S., Miettinen, M., Dang, M. H., Asokan, N., and Sadeghi, A. Dïot: A crowdsourced self-learning approach for detecting compromised IoT devices. *CoRR abs/1804.07474* (2018).

[59] Openwrt. Openwrt project. https://openwrt.org/.

[60] OVH. OVH news - the DDoS that didn't break the camel's VAC. https://www.ovh.com/us/news/articles/a2367.the-ddos-that-didnt-break-the-camels-vac.

[61] Özçelik, M., Chalabianloo, N., and Gur, G. Software-defined edge defense against IoT-based DDoS. In *2017 IEEE International Conference on Computer and Information Technology (CIT)* (Aug 2017), pp. 308–313.

[62] Piedrahita, A. F. M., Rueda, S., Mattos, D. M. F., and Duarte, O. C. M. B. Flowfence: a denial of service defense system for software defined networking. In *Global Information Infrastructure and Networking Symposium (GIIS)* (Oct 2015), pp. 1–6.

[63] Rangasamy, J., Stebila, D., Boyd, C., and Nieto, J. G. An integrated approach to cryptographic mitigation of denial-of-service attacks. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security* (New York, NY, USA, 2011), ASIACCS '11, ACM, pp. 114–123.

[64] Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., and Knightly, E. DDoS-Shield: DDoS-resilient scheduling to counter application layer attacks. *IEEE/ACM Transactions on Networking 17*, 1 (Feb 2009), 26–39.

[65] Rashidi, B., Fung, C., and Bertino, E. A collaborative DDoS defence framework using network function virtualization. *IEEE Transactions on Information Forensics and Security 12*, 10 (Oct 2017), 2483–2497.

[66] RESEARCH, T. M. MQTT and CoAP: Security and privacy issues in IoT and IIoT communication protocols. https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/mqtt-and-coap-security-and-privacy-issues-in-iot-and-iiot-communication-protocols.

[67] ROMERO, J. M. Hackers broadcast live stream of police camera at Podemos leaders' home. https://elpais.com/elpais/2019/04/08/inenglish/1554706393_909358.html.

[68] Schneier, B. The Internet of Things is wildly insecure—and often unpatchable. https://www.schneier.com/essays/archives/2014/01/the_internet_of_thin.html.

[69] Shodan. Shodan search engine front page. https://www.shodan.io/.

[70] Sivanathan, A., Gharakheili, H. H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A., and Sivaraman, V. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing 18*, 8 (Aug 2019), 1745–1759.

[71] Sivanathan, A., Sherratt, D., Gharakheili, H. H., Radford, A., Wijenayake, C., Vishwanath, A., and Sivaraman, V. Characterizing and classifying IoT traffic in smart cities and campuses. In *Proceedings of the IEEE Infocom Workshop on Smart Cities and Urban Computing* (May 2017), pp. 559–564.

[72] Sommer, R., and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2010), SP '10, IEEE Computer Society, pp. 305–316.

[73] Spring, T. Smart lock turns out to be not so smart, or secure. https://threatpost.com/smart-lock-turns-out-to-be-not-so-smart-or-secure/146091/.

[74] Torabi, S., Bou-Harb, E., Assi, C., Galluscio, M., Boukhtouta, A., and Debbabi, M. Inferring, characterizing, and investigating Internet-scale malicious IoT device activities: A network telescope perspective. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (June 2018), pp. 562–573.

[75] Wang, H., Zhang, D., and Shin, K. G. Detecting SYN flooding attacks. In *Joint Conference of the IEEE Computer and Communications Societies* (June 2002), vol. 3, pp. 1530–1539.

[76] Wang, R., Jia, Z., and Ju, L. An entropy-based distributed DDoS detection mechanism in software-defined networking. In *IEEE Trustcom/BigDataSE/ISPA* (Aug 2015), vol. 1, pp. 310–317.

[77] Webe, D. Why it's so hard to implement IoT security. https://www.securityweek.com/why-its-so-hard-implement-iot-security.

[78] Wood, P., Gutierrez, C., and Bagchi, S. Denial of Service Elusion (DoSE): Keeping clients connected for less. In *IEEE 34th Symposium on Reliable Distributed Systems (SRDS)* (Sep. 2015), pp. 94–103.

[79] Xerox, C. 5 reasons why IoT security is difficult. https://www.xerox.com/en-us/insights/iot-security.

[80] Zaalouk, A., Khondoker, R., Marx, R., and Bayarou, K. OrchSec: an orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions. In *IEEE Network Operations and Management Symposium (NOMS)* (May 2014), pp. 1–9.

[81] Zhang, S. Creepy website is streaming from 73,000 private security cameras. https://gizmodo.com/a-creepy-website-is-streaming-from-73-000-private-secur-1655653510.