Standardizing IoT Network Security Policy Enforcement

David Barrera
Polytechnique Montréal
david.barrera@polymtl.ca

Ian Molloy
IBM T.J. Watson Research Center
molloyim@us.ibm.com

Heqing Huang
IBM T.J. Watson Research Center
hhuang@us.ibm.com

***Abstract*—This paper discusses the design of a standardized network security policy enforcement architecture for IoT devices. We show that the network behavior of many consumer IoT devices tends to be predictable, and thus simple to profile and constrain. We propose an automated approach to derive network security policies for devices without requiring vendor cooperation or changes to devices or cloud infrastructure. We also describe a scalable and effective multi-layered policy enforcement architecture that limits the impact of IoT device compromise.**

I. Introduction

The rise in attacks targeting IoT devices (e.g., Mirai [5] or Brickerbot [17]) is perhaps unsurprising. As the cost of adding wireless capabilities to devices decreases, more consumer electronics, toys, appliances, and other "things" have become Internet-enabled. A 2015 report [8] estimates that by 2020, there will be around 20 billion IoT devices online, with 65% of those existing in the consumer space. IoT devices exhibit desirable characteristics for attackers: they are powered-on and networked 24/7, they use weak security configurations, and they are underpowered and thus unable to run anti-malware, intrusion detection, or auditing services. Software update procedures for IoT devices tends to be clunky and error-prone, allowing devices to remain unpatched for long periods of time. Moreover, embedded devices often lack displays, making it difficult to know whether the device is behaving as expected at any given time.

When vulnerabilities are discovered in IoT products, some vendors are quick to develop and distribute patches. However, this level of support is currently far from the norm, with many vendors discontinuing products support shortly after product release, leaving user devices and data at risk. Proposals for securing IoT devices, including best practices [15], secure hardware [13], secure operating systems [14], and secure protocols [9] exist, but are not yet widely deployed. While we wait for these proposals become standard, end users and network administrators need immediately deployable tools to help secure their IoT devices.

One strategy to reduce the impact of IoT device compromises is to enforce the principle of least privilege for every device on the network. By restricting what a device can connect to and what type of data it can send out, malicious activity originating on the device can be blocked without changes to hardware or software. To achieve this, we propose the creation and use of standardized network security policies that describe the essential network behavior of IoT devices. For example, a smart light bulb should accept on/off commands from its manufacturer's cloud servers, but should never send HTTP traffic to Twitter or Reddit. Given the small network footprint of many devices, these types of network security policies could be automatically learned by observing a device. Alternatively, policies could be provided by manufacturers or retrieved from sources curated by experts.

This paper discusses the motivation and requirements for IoT network security policies, and demonstrates their use on a small network. We focus on standardizing the policies instead of the enforcement for several reasons. (1) It encourages a more decentralized ecosystem where many vendors can offer software and appliances to enforce the network access rules. (2) It allows policy enforcement to be deployed at gateway routers, smartphones, middleboxes, IoT hubs, etc. (3) Policies can be audited and refined, giving visibility and transparency into device behavior.

II. Background and Related Work

This section briefly describes relevant IoT security research. A more detailed treatment including surveys, regulatory approaches, and new architectures is given our technical report [3]

Large-scale analysis. A 2010 study [6] scanned the IPv4 address space and found about 13% of devices that responded were vulnerable to compromise due to the use of default credentials. Costin et al. [2] analyzed 32,000 embedded device firmware images for printers, routers, cameras, etc. Without prior knowledge of the

Workshop on Decentralized IoT Security and Standards (DISS) 2018
18 February 2018, San Diego, CA, USA
ISBN 1-891562-51-7
https://dx.doi.org/10.14722/diss.2018.23007
www.ndss-symposium.org

firmware image layout or access to the device for which the firmware was developed, the authors were able to extract 35,000 RSA private keys, weak password hashes, and hardcoded credentials.

Device-specific attacks. Many IoT devices have been found to be insecure. Ho et al. [11] evaluated the security of several IoT smart locks, finding that cloud-based access control rules are a poor choice for locks due to the potential lock-outs if the cloud services is unavailable. A survey on printer security [12] found the presence of decades-old vulnerabilities in new printers. Ronen et al. [18] identified critical vulnerabilities in Philips smart lightbulbs which allowed the authors to recover the private key used to authenticate firmware updates.

Inter-IoT Security Policies. Most closely related to our present focus, Yu et al. [20] proposed a software-defined networking (SDN) architecture to secure inter-IoT device communication. Their *IoTSec* proposal adds a virtual middlebox between each IoT device on the network and the gateway. At each middlebox, a high-level policy (i.e., defining allowed application-layer interactions rather than packets or protocols as described herein) is installed, which defines a set of allowed interactions between a protected device and other devices on the network. IoTSec requires that home networks be re-architected to support SDN, and that all possible cross-device interactions be enumerated in order to create the policies.

III. Toward Standard Network Security Policies for IoT Devices

Today's consumer IoT devices do not behave like general purpose computers. The lack of graphical or other interfaces on most IoT devices prevents users from directly running their own software on the devices, even though the underlying operating system may support it. However, many networks treat IoT devices as general purpose computing systems, giving the IoT devices unrestricted network access. This is despite the fact that IoT devices typically only require a small set of connections to support their functionality. This over-privilege creates an opportunity for attackers to use victim IoT devices to launch other attacks (e.g., the Mirai DDoS attacks [5]).

Table I summarizes the network behavior of 19 IoT devices. To populate this table, we monitored network traffic to/from each device for approximately 12 minutes beginning with device power-on. We avoided interacting with devices during the capture period, which gave us a baseline of network activity performed automatically by devices at boot time. We augmented our dataset with the public data of Sivanathan et al. [19]. Our analysis shows that simple devices such as digital scales, smart light bulbs, and Bluray players have a small network footprint. These devices look up a small set of domain names (typically API endpoints and domains used for network connectivity checks), and only connect to the servers returned by the DNS lookups of those domain names. By contrast, more complex devices allowing installation of apps (e.g., laptops, mobile phones, and game consoles) connect to a larger set of remote hosts and perform more DNS lookups.

Device	Distinct Endpoints	Distinct Domains	HC IPs
AT&T Microcell	2	0	2
Fitbit Aria Digital Scale	2	1	0
Withings Smart scale†	2	1	0
Withings Baby Monitor†	2	1	0
PIX-STAR Photo-frame†	2	1	0
Belkin Wemo switch†	2	1	0
Blipcare BP meter†	2	1	0
Samsung Bluray Player	4	1	0
Netatmo Weather Station	5	1	0
LIFX Gen 1 bulb✳	5	1	0
LIFX Gen 2 bulb✳	5	2	0
Triby Speaker†	6	2	0
NEST Smoke Alarm†	6	4	0
TP-Link Smart plug†	7	2	0
Netatmo Welcome†	7	2	6
Amazon Fire TV	8	4	0
Amazon Kindle	9	8	1
TP-Link Cloud camera†	15	2	3
Amazon Echo✳	20	13	0
AppleTV 4th Gen	37	23	2
Samsung Galaxy Tab†✳	48	21	0
Android Phone†	57	48	0
Microsoft XBox One	74	57	0
Laptop†	140	101	0

TABLE I: Network behavior of several IoT devices. General purpose computing systems given in the bottom rows for comparison. HC IPs are hardcoded IP addresses, marked if no corresponding DNS lookup was made prior to connecting to an IP address. Devices with ✳ ignored the DHCP-provided DNS resolver and used Google's resolver (8.8.8.8) instead. Data for devices with † was obtained from the public dataset of Sivanathan et al. [19].

A. Design Goals

In designing network security policies, our goal is to precisely describe the behavior of a given IoT devices that is required for regular operation. The policies specify network activity that should be allowed, while any traffic not specified in the policy is dropped at a policy enforcement point. Our focus on standardizing the policies, instead of the enforcement points or mechanisms, seeks to achieve the following goals:

- **Interoperability.** Standard network security policies enable a diverse set of policy enforcement points and mechanisms. This gives users freedom to select an enforcement architecture that is suitable to their environment and devices. For example, some home users may be unable to replace their ISP-provided router/modem. Other users may have Bluetooth-only devices, necessitating enforcement on a mobile phone or tablet.

- **Deployability.** Policies should be easily deployable by not requiring changes to either the IoT devices or to the cloud services that that support them. The policies should be modifiable without requiring vendor or third-party support (but could benefit from such support, as explained later).
- **Extensibility.** With the growing applications of IoT, it is unreasonable to expect a solution designed for today's devices will also work for all future devices. The policies should be extensible to support new behavior as devices and technologies evolve. Similarly, the enforcement points can be swapped out for more powerful or more effective alternatives when available.

B. Policy Details

Our proposed IoT security policies are machine-readable descriptions of expected network behavior for the IoT device. Policies are whitelists, meaning that any outgoing traffic that is not defined in the policy will be denied. We chose a whitelisting approach instead of blacklisting for two reasons. First, whitelisting, when describing a narrow set of behavior, provides the strongest security guarantees; it forces an adversary to operate within the confines of rules in the whitelist, as opposed to operating around rules of a blacklist. Second, IoT vendors designing devices should be able to describe how and to what the device needs to connect; while IoT developers may not be security experts, they must be aware of network activity since it is this very activity that gives the device functionality. Because of the whitelisting approach, device policies must ensure the inclusion of rules for all expected connections including periodic API calls, user-triggered network behavior, software/firmware updates, etc.

Listing 1: Netatmo weather station sample policy

```
1  {"Netatmo Weather Station": {
2    "MACAddr": "70:ee:50:13:ab:cd",
3    "IPAddr": "172.16.1.2",
4    "AllowedDNSQueries": [
5      {"type": "A", "query": "netcom.netatmo.net
         ", "resolver": "192.168.1.1"}
6    ],
7    "AllowedDNSReplies": [
8      {"type": "A", "query": "netcom.netatmo.net
         ", "answers": "62.210.92.0/24"}
9    ],
10   "AllowedConnections": [
11     {"family": "IPv4", "dest": "netcom.netatmo
         .net", "proto": "TCP", "dstport": "2505
         0", "freq": "6/hr"}
12   ]}}
```

Listing 1 shows an example policy for the Netatmo weather station. Our analysis of network traces collected for the weather station revealed that the device wakes up every 10 minutes, performs readings of CO_2, temperature, air quality, and air pressure, and uploads the measurements to Netatmo's cloud servers. To obtain the IP address(es) of Netatmo's servers, the device performs an IPv4 (type A) DNS lookup of `netcom.netatmo.net`. The upload takes place over TCP on port 25050 to an IP address returned by the previous DNS lookup. Line 11 in Listing 1 concisely captures all the described behavior. It allows outgoing IPv4 TCP connections to port 25050 to any IP address returned by a lookup to `netcom.netatmo.net`, with at most 6 of these connections being initiated per hour (one every 10 minutes). Line 8 restricts the IP addresses that are allowed as answers when performing the DNS lookup, and Line 5 allows lookups of only one domain name via a single resolver.

Type	Example Parameters
Metadata	Schedule, rate, bandwidth, packet size
Contents	Protocol, IP Address, port number, connection flags/state
Application	Types of DNS lookups and responses, TLS certificates, HTTP GET/POST/PUT request

TABLE II: Example parameters that could be defined in the network security policies

Note that the minimal policy in Listing 1 appears to be sufficiently restrictive. However, even such a policy could leave room for an attacker to be disruptive. For example, the policy would permit an attacker gaining control of this Netatmo weather station to flood the DNS resolver with a large number of A lookups of `netcom.netatmo.net`, or send gigabytes of TCP traffic to any of Netatmo's servers. The policy could be further tightened by specifying additional restrictions such as number of bytes, packets, or number of allowed lookups. Table II shows additional options that could be defined in the policy. While the table is not meant to be comprehensive, we note that adding a new parameter to the policy only requires a corresponding way to inspect and enforce that parameter at the enforcement point.

C. Obtaining policies

We envision several ways to obtain a policy for a given device.

1) **Manufacturer.** The device manufacturer can create the policies for devices they ship. We believe manufacturers are in the best position to do so, since they also develop or commission the software for the device. It is thus reasonable to expect the manufacturers to know what functionality the device needs. Policies could be made available through vendor websites (e.g., a QR code on the box pointing to mysmarttoaster.io/securitypolicy), or distributed along with the software for managing the device.
2) **Third party.** Policies can be written by third parties, either from scratch after analyzing the device's

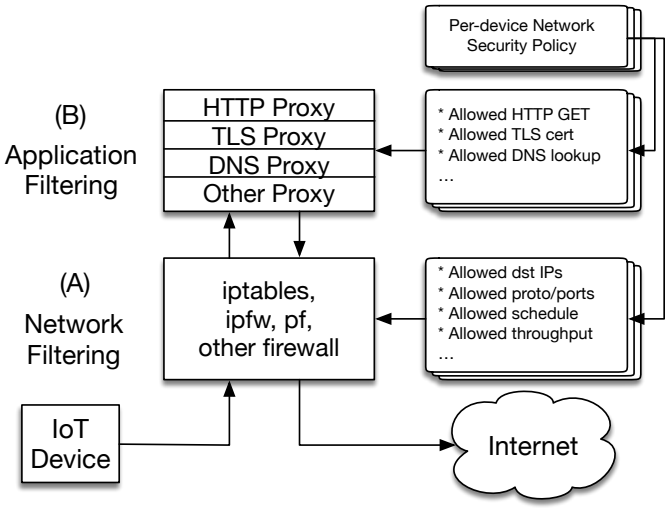

Fig. 1: Components of the network security policy enforcement architecture.

behavior, or by modifying manufacturer-provided policies to be more/less restrictive (e.g., by removing/adding rules). The home automation community has already enabled integration of vendor-unsupported services and devices (e.g., Homebridge[1], Home Assistant[2]), so they may provide policies for certain devices. Anti-malware and security firms could also provide policies as a service to their customers, creating an additional revenue stream.

3) **Automatic.** It is possible to programatically create policies by observing the network behavior of a device for a given amount of time. The device can be assigned a temporary *allow all* rule at the enforcement point, during which all network traffic is recorded. After the monitoring period ends, a policy matching the observed behavior can be created and enforced.

Once a policy has been retrieved and is being enforced, functionality changes to the behavior of an IoT device (e.g., through a firmware update) may require updating the previously installed policy. We expect that policy enforcement points will require a mechanism to securely authenticate and verify updated policies. One strategy may be to digitally sign policies and verify the validity of the signature against a set of pre-installed trust roots. Alternatively, self-signed certificates along with a trust-on-first-use mechanism (à la Android [4]) could be used. We leave policy verification and updates to future work.

D. Policy Enforcement

To be effective at restricting network capabilities, the policy enforcement logic must be positioned at a vantage point where all network traffic to and from IoT devices can be inspected. Such vantage points will vary depending on the wireless technology used by the IoT devices, and depending on the topology of the network being instrumented. For example, devices using Zigbee, Bluetooth, and similar short-range network technologies require a hub or smartphone to provide connectivity to cloud services. For these devices, policy enforcement can be built into the hub if supported, or the hub itself may be treated as an Internet-capable IoT device. For devices that support IP connectivity, enforcement can be applied at the wireless access point, WAN gateway, or at a middlebox between the wireless AP and the gateway.

Once an enforcement point has been selected, a policy obtained through one of the methods described in Section III-C is installed. Depending on the type of entries in the policy, rules are loaded into one of two filtering modules, as shown in Figure 1. Network layer and metadata entries are loaded into the network filtering module (A). This module converts entries into software firewall rules suitable for use in one of the well-known packet filtering frameworks on Linux, BSD, or other firewall appliances. Application layer entries are loaded into application-layer proxies, each implementing their own enforcement logic. Each proxy inspects payload data from packets, and transparently forwards the request to the destination if the request is compliant with the policy.

E. Auditability

An inevitable consequence of the large number of IoT devices is that non-expert users will become the administrators of dozens of devices. We see IoT network security policies as a step toward improving the auditability of devices, since its machine readable policies can easily be converted to human readable form and displayed on another device (e.g., smartphones or PCs). These policies could allow even non-experts to gain visibility into what their devices are allowed to do. For example, the entries `max-bw-out: 10M/w` and `valid-domains: api.lightbulbs.io` can be converted to: *"This light bulb will not send more than 10 MB of data per week to api.lightbulbs.io"* or simpler *"This light bulb will only send data to api.lightbulbs.io"*.

Another way to offer transparency and auditability is to collect and display statistics at the policy enforcement point. Measurements showing number of times a rule has been matched, or displaying extraneous connections can help identify devices that are misbehaving or policies that are too restrictive.

IV. Proof of concept

As a proof of concept, we implemented a custom policy enforcement system and created policies for 3 devices: a Netatmo Weather station, a LIFX smart light bulb, and a Fitbit Aria digital scale.

[1] https://github.com/nfarina/homebridge
[2] https://home-assistant.io

Our test environment was a Raspberry Pi Zero W running a recent build of Archlinux. The Raspberry Pi acts as a middlebox between IoT devices and the network gateway by advertising a wireless network using `hostapd`[3]. We configured link layer isolation by setting `ap_isolate=1` in the `hostapd` configuration. This prevents wireless clients from seeing each other at layer 2. We selected `iptables`, Linux's built-in packet filtering framework to enforce network layer rules. For DNS filtering, we selected `dnsmasq` [7], a lightweight network infrastructure tool that supports DNS forwarding.

A. Creating Policies

For each device, we created a policy using the automatic method (see Section III-C) and manually inspected the policies for correctness. To automatically create the policies, we developed a tool in `golang` that reads network packet captures (in the form of pcap files) and produces a JSON policy as shown in Listing 1.

For the weather station and the light bulb, we captured network traffic for 12 minutes. Both devices performed some background activity. The digital scale only performs network activity for uploading measurements. Thus, we stepped on the scale twice and recorded network activity.

Once the packet capture has been read, the tool removes duplicate entries (e.g., recurring DNS queries or TCP connections) and produces a policy object. The object can then be printed to the screen, or written to a file in JSON or other formats.

B. Enforcement

We wrote a simple tool that takes as input a policy file and converts entries into `iptables` rules or `dnsmasq` whitelist entries. For example Listing 1 is converted to:

```
#iptables -t nat -A PREROUTING -i wlan0 \\
-s 172.16.1.2 -d 62.210.92.0/24 -p tcp \\
--dport 25050 -m limit --limit 6/hour -j ACCEPT

#iptables -t nat -A PREROUTING -i wlan0 \\
-s 172.16.1.2 -d 192.168.1.1 -p udp \\
--dport 53 -j ACCEPT
```

Note that our tool combines multiple policy entries (in this case lines 7 and 11) to create a stricter `iptables` rule. Since the connections entry specifies a destination hostname, and there is a corresponding rule specifying allowed IPs for that hostname, the rule can precisely specify allowed sources (-s) and destinations (-d). The second rule allows UDP traffic to destination port 53 as required for DNS lookups. The firewall is configured to drop all traffic that doesn't match at least one rule, and to allow replies to connections that were allowed outbound.

We configured `dnsmasq` to only forward received DNS queries that are allowed by the policy using `no-resolv` and by whitelisting domains. Whitelisted entries are extracted from the JSON policies by looking for the "AllowedLookups" directive. Allowed lookups are added to the `dnsmasq` configuration file as: `server=/netcom.netatmo.net/8.8.8.8`. Any lookup that isn't whitelisted will receive an answer of 127.0.0.1 (`address=/#/127.0.0.1`).

C. Testing

After deploying the enforcement rules, we attempted to use the IoT devices to ensure their functionality was not impaired by our filtering. The Fitbit Aria successfully uploaded weights. The Netatmo weather station was able to perform periodic reporting every 10 minutes, but repeated on-demand readings (triggered by pressing a button on top of the device) made the number of connections exceed the 6/hour threshold. A more permissive value of 10-20 per hour may be more appropriate to allow a small number of on-demand readings.

The LIFX bulb worked as expected, although with higher latency between commands and responses. The LIFX bulb can be controlled through a smartphone application, which is expected to be on the same local network as the bulb. By having the bulb and smartphone on different networks, commands were sent to LIFX's cloud servers, which were then read by the bulb's long-lived TCP connection to the same servers. While this added latency, it had no effect on functionality.

V. Discussion

This section discusses technical challenges in deploying network filtering solutions to secure IoT devices.

Device-to-device connectivity. Certain IoT devices require discovery and connectivity to other devices on the local network. In particular, devices that do not rely on cloud services may operate by discovering nearby devices and interacting with them directly. While disabling access point isolation may enable certain use cases, it also opens up the all devices on the wireless network to attacks. There may be opportunities for "selective AP isolation", where devices are allowed to communicate only with authorized devices on the same network.

Device identification. Current IP networks identify devices based on layer 2 identifiers (MAC addresses) and IP addresses. When creating or loading a policy that applies to a given device, it is still possible for a compromised device to modify its behavior, and simultaneously modify its identifiers. Miettinen et al. [16] show that fingerprinting device types can be done

[3] https://w1.fi/hostapd/

with high accuracy, but identifying distinct firmware builds or hardware variants of the same device is more challenging. Being unable to identify a device correctly could allow a device to spoof the behavior of a different device with a less restrictive policy. While better fingerprinting techniques are developed, an alternative solution to this problem is remote attestation, but this requires a trusted hardware module.

Complex IoT devices. Throughout the paper, we have described how our proposed policy enforcement framework can be effective when devices have a small set of predictable functionality. Given the rapid pace of innovation in IoT, it is reasonable to expect IoT devices to grow in complexity. As devices gain features that allow customization or extensibility, our ability to profile and restrict them drops. This is already the case for IoT-ish devices like the Xbox or the AppleTV (see Table I). These multimedia boxes allow the installation of applications, blurring the line between single-purpose functionality and general purpose computers. Because each new application may require connecting to a variety of cloud services, enumerating all possible servers and protocols may become infeasible. Personal desktop firewalls experienced usability challenges for this reason over a decade ago; repeated prompts to allow network connectivity for each new application were often dealt with by allowing all outbound connections [10].

WAN-enabled IoT devices. As wireless technology costs decrease, manufacturers may start shipping products with built-in WAN connectivity. Direct WAN connectivity increases usability by removing the need for complex network attachment procedures, and also gives vendors direct access to the device for diagnostics and updates. The downside of direct WAN connectivity is the consumer's inability to control the communication channel. IoT devices with such capabilities already exist; for example, the Amazon Kindle can download books and updates over its built-in 3G connection. Another emerging technology is LoRaWAN [1], a low-power wireless protocol which allows devices to effortlessly join city-wide networks.

VI. Conclusion

This paper discussed the challenges in securing billions of consumer IoT devices. We argue that security solutions requiring vendor involvement, such as modifications to hardware and software are unlikely to be successful. Instead, we show that IoT devices have a small network footprint which facilitates the creation of standard network security policies. These policies help enforce the principle of least privilege on IoT networks and reduce the impact of device compromise even on unsupported devices.

References

[1] L. Alliance, "LoRaWAN Home," 2016, accessed Apr 14, 2017. [Online]. Available: https://www.lora-alliance.org

[2] Andrei Costin, Jonas Zaddach, Aurelien Francillon, and Davide Balzarotti, "A Large-Scale Analysis of the Security of Embedded Firmwares," in *USENIX Security*, 2014.

[3] D. Barrera, I. Molloy, and H. Huang, "IDIoT: Securing the Internet of Things like it's 1994," *ArXiv e-print 1712.03623*, Dec. 2017.

[4] D. Barrera, J. Clark, D. McCarney, and P. van Oorschot, "Understanding and Improving App Installation Security Mechanisms through Empirical Analysis of Android," in *ACM SPSM*, 2012.

[5] I. Z. Ben Herzberg, Dima Bekerman, "Breaking Down Mirai: An IoT DDoS Botnet Analysis," 2016, accessed Apr 14, 2017. [Online]. Available: https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html

[6] A. Cui and S. J. Stolfo, "A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan," in *ACSAC*, 2010.

[7] Dnsmasq, "Dnsmasq," 2017, accessed Apr 14, 2017. [Online]. Available: http://www.thekelleys.org.uk/dnsmasq/doc.html

[8] Gartner, "Gartner Says 6.4 Billion Connected Things Will Be in Use in 2016, Up 30 Percent From 2015," 2015, accessed Apr 14, 2017. [Online]. Available: http://www.gartner.com/newsroom/id/3165317

[9] J. Granjal, E. Monteiro, and J. Sa Silva, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1294–1312, 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7005393/

[10] A. Herzog and N. Shahmehri, "Usability and Security of Personal Firewalls," in *New Approaches for Security, Privacy and Trust in Complex Environments*, 2007.

[11] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart Locks: Lessons for Securing Commodity Internet of Things Devices," in *ACM CCS*, 2016.

[12] Jens Müller, Vladislav Mladenov, and Juraj Somorovsky, "SoK: Exploiting Network Printers," in *IEEE Security and Privacy*, 2017.

[13] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens, "Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base," in *USENIX Security*, 2013.

[14] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish *et al.*, "sel4: Formal verification of an os kernel," in *ACM SIGOPS*, 2009.

[15] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004.

[16] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "IoT Sentinel: Automated Device-Type Identification for Security Enforcement in IoT," in *IEEE Distributed Computing Systems (ICDCS)*, 2017.

[17] Radware, "BrickerBot Results In PDoS Attack," 2017, accessed Apr 14, 2017. [Online]. Available: https://security.radware.com/ddos-threats-attacks/brickerbot-pdos-permanent-denial-of-service

[18] E. Ronen, C. O'Flynn, A. Shamir, and A.-O. Weingarten, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction," in *IEEE Security and Privacy*, 2017.

[19] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and Classifying IoT Traffic in Smart Cities and Campuses," in *IEEE Infocom Workshop on Smart Cities and Urban Computing*, May 2017.

[20] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things," in *ACM HotNets*, 2015.