HANZO: Collaborative Network Defense for Connected Things

Aman Singh *, Shashank Murali *, Lalka Rieger ‡, Ruoyu Li ‡
Stefan Hommes †, Radu State †, Gaston Ormazabal ‡ and Henning Schulzrinne ‡
* Palindrome Technologies
Email: {aman.singh, shashank.murali}@palindrometech.com
† Center for Security, Reliability and Trust (SnT), University of Luxembourg
Email: {stefan.hommes, radu.state}@uni.lu
‡ Department of Computer Science, Columbia University
Email: {ler2161, rl2929}@columbia.edu, {gso, hgs}@cs.columbia.edu

***Abstract*—The IoT devices are typically shipped with default insecure configurations and vulnerable software stacks rendering host networks exposed to attacks, especially small networks with no administration. We present a network system model for device configuration and operations management. Using this model, we design and implement an autonomous network management platform with device classification and traffic characterization functions integrated in a network gateway. We evaluate the system using a connected home testbed that combines IoT and general-purpose devices.**

I. Introduction

The Internet of Things (IoT) enable integration of the physical with the digital world resulting in process automation and optimizations. In general, an IoT system contains three different phases - collection, transmission, and processing. The collection phase captures system events, the transmission phase communicates the events to a system controller, and the processing phase generates system output actions based upon system input events. While IoT devices have limited functionality, they still increase the attack surface as most of these devices have vulnerable firmware, insecure default passwords, and no security management functions. A compromised IoT system can be used for DDoS attacks [1], digital crime proxies [2], ransomware [3], cryptocurrency malwares and data theft [4].

There are enterprise-centric technical solutions for securing IoT networks that require proper network administration. A significant challenge is to apply security controls effectively in consumer-centric devices that are deployed in home or small networks with no trained network administrators. Since IoT devices are built to provide only limited functionalities, their communication behavior is more restricted, and thus the typical communication endpoints can be easily derived from network traffic. Furthermore, the device manufacturers know the external endpoint configuration *a priori* because they control device functions, e.g., service endpoints such as software updates and event notification. This information can be derived deterministically by the manufacturer, or by any third-party that can assess the device operational parameters. Once these configuration parameters are derived for each device they can be shared with other network installations enabling a collaborative configuration sharing environment.

978-1-5386-6205-2/18/$31.00

The Manufacturer Usage Descriptions (MUD) specification [5] leverages the manufacturer's knowledge of the device operations to create a device profile using Yet Another Next Generation (YANG) data modeling language [6]. The MUD profile can provide signals to a network controller about connectivity a device requires to properly function. The MUD profile includes a network-acl-model [7] and some additional metadata related to device operations. In this way, a network can define device specific policies to prevent communication attempts to endpoints that are outside of the scope of the device's functionality. Figure 1 shows a device with MUD support in a network. The MUD-URL emitted from the device identifies the profile location. Subsequently, a new network element, the MUD manager, downloads the profile from the manufacturer file server, and installs it locally. The MUD proposal requires device manufacturer support along with IoT device support and integration of MUD manager in the network environment.

We present a network system model that provides device configuration, security and operations management. Using this model, we design and implement the Home Area Network Zero Operations (HANZO) controller. The controller manages IoT devices and generates dynamic device profiles, HANZO profiles, that are similar to MUD profiles. We show that all required MUD profile parameters can be derived from network traffic observation. The HANZO profile includes communication endpoints and device identification fingerprint data. The endpoints are derived using traffic observation time windows. The device fingerprinting is derived using a combination of protocol heuristics and traffic classification methods. The HANZO profiles are converted into network-supported ACLs that limit device communication endpoints. Subsequently, we implement ACLs using an SDN infrastructure for better programmability by directly mapping ACLs into flow-table rules. The proposed architecture is suitable for deployment in small networks such as a home or small office buildings. We evaluate the system using a model connected-home testbed containing

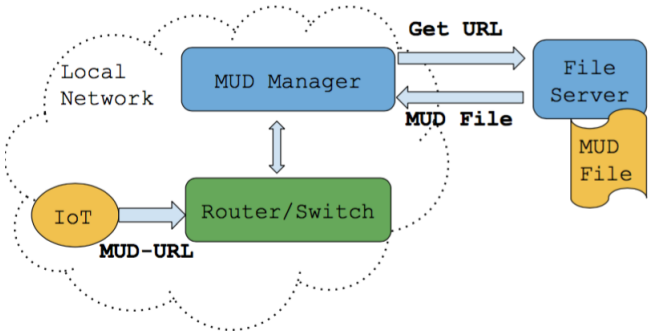

Fig. 1: Manufacturer Usage Description (MUD) flow

both IoT and general-purpose devices.

The remainder of this paper is structured as follows: Section II describes IoT device security management problem background. Section III describes a IoT network system model including security trust model. Section IV describes the system architecture along with a connected-home testbed description. Section V describes implementation details. Section VI describes evaluation of the architecture against the connected-home testbed. Section VII describes recent related work in automated IoT device security methods. Finally, Section VIII concludes along with future work plans.

II. Background

The IoT security and privacy threats have been well documented [8]. Any adversary can search for vulnerable devices using security search engines such as shodan [9] and censys [10]. The network worm frameworks such as BASHLITE [11] and the more recent Mirai [1] search for vulnerable devices ranging from home network devices such as DVRs, IP cameras, routers and printers, to city network devices such as traffic lights and environment sensors. These worms install malware on vulnerable devices, converting them into remote bots that are used as part of large-scale botnet attacks. Furthermore, the public disclosure of the Mirai source code has already generated more complex worm strains. On the opposite side, white-hat worms such as Hajime [12] and BrickerBot [13] are attempting to use the same attack patterns to disable IoT devices after successful exploits.

Enterprise networks demand security baseline controls and network operations administration. The enterprise IoT platforms further provide granular device security controls and vendor supply-chain management that further hardens the oeprations. Lack of network administration knowledge and cost can render these management platforms ineffective for small network installations such as homes and buildings.

An IoT device usually follows - Registration, Configuration, Operation, Maintenance and Decommission steps during its life-cycle. For seamless user experience and secure IoT ecosystem, automated device identification and authentication are critical. The public key certificates provide secure device identities. During registration step, a device can share identity and perform authentication with a network - before, during or after the IP address acquisition process. The Extensible Authentication Protocol over LAN (EAPoL) [14] with a X.509 certificate enables layer 2 authentication before IP address acquisition. The DHCP Auth header [15] enables authentication during IP address acquisition. An application layer authentication service, e.g. IoT auth framework [16] enables device authentication after IP address acquisition. The MUD architecture proposes extensions to Link Layer Discovery Protocol (LLDP), X.509 certificates and DHCP Options header for device identification and profile download using MUD-URL string [5]. In contrast, small networks typically have limited or no authentication infrastructure setup besides baseline network password authentication. As MUD architecture requires device changes for MUD-URL string, for current IoT devices, the HANZO controller applies the same MUD principle of restricting IoT device communication endpoints by observing device network traffic.

III. System Model

An IoT network can be modeled as a network of networks with sensors, controllers and actuators. For example, an electrical network can be monitored with smart meters and controlled with smart switches. The networks can be categorized as lights, temperature, water, gas, appliances, presence, media/entertainment, waste management etc. Similar to a graph, we can model an IoT network with the following entities - ***Device, Profile, Edge*** . A device (*D*) can be a sensor or an actuator. The device-to-device interaction creates an edge (*E*). An edge has two endpoints, one for each device. A profile (*P*) defines communication constraints for a device, i.e. set of edges that are allowed. The composition of **{D, P}** forms a network (*N*). The network is a composite entity in the system, i.e., the network state is dependent on other entity values.

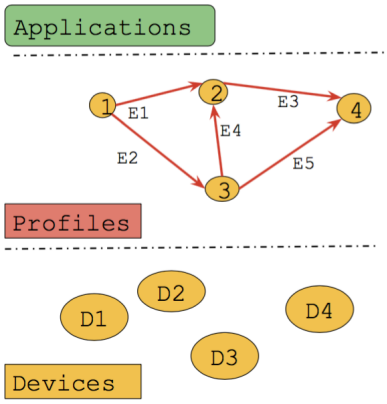

Fig. 2: System model

Once the device profiles are enforced, applications can only communicate via allowed edges in the network. Figure 2 shows the layered composition of network with profiles applied on devices and application contrained by profiles. Each device has an associated device management application (*M*). It is an essential entity that manages the life-cycle functions of

a device - identity certificates, configuration, firmware updates, and data storage models. These functions can be managed by the device manufacturer itself or by a different third-party application management platform.

A. Trust Model

A device (D_i) is managed by an application (M_i). Each device is identified by a certificate issued by the manufacturer. All D_i trust the M_i. If a network (N_i) trusts a M_i then all the D_i issued by M_i are also trusted. The trust between N_i and M_i can be established by leveraging current public Certificate Authority (CA) based systems. For devices with no verifiable identity, N_i marks the device as untrusted. For example, figure 3 shows manufacturers, M1 and M2, establish trust with the CA in Step 1. Step 2 creates trust between the network controller and the CA. Both manufacturers can issue devices with valid identities that can be verified and trusted by the network controller in Step 3. Furthermore, the device M'-D1 is untrusted because of no valid identity.

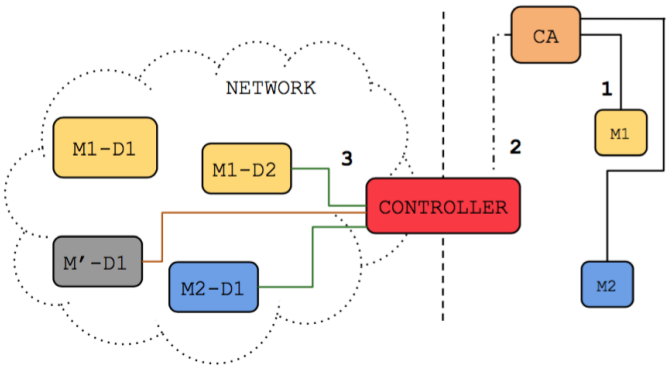

Fig. 3: System trust model

B. Device Model

The device is modeled with a network-centric view of different life-cycle states.

Registration - In this state, the device has booted up successfully and sends its identification, if configured, to the network controller. The network controller can perform device authentication. The authentication process involves device identity verification and establishing trust between device and network. The device is tagged as "Trusted" or "Untrusted" after authentication step. The device acquires a network IP address after successful registration.

Configuration - After successful registration, the network controller applies a device profile. The profile contains operational parameters such as communication endpoints for device life-cycle management and service functions. The profile can be generated by the manufacturer, network admin or generated dynamically by observing initial device traffic.

Operation - After successful device configuration, the device becomes operational in the network. It is allowed to send communication packets to other devices in the network.

Maintenance - In this state, the device performs firmware updates, or any configuration changes. The device management application triggers the transition to this state. After a successful firmware update process, the device again goes through registration and configuration states.

Quarantine - During operation state, if the network controller detects the device is compromised, it is isolated from other devices, by blacklisting the device profile and blocking all communication links. Furthermore, the controller can notify any connected management applications. Based upon management application input, the device can either go back to operation state, or maintenance state, or be removed from the network. If the device is infected, and an update process may not fully recover it, it is advisable to discard it.

C. Edge Model

The device-to-device communications are modeled as graph edges. The edges are directional and defined using the 5 parameter tuple [src-endpoint, src-port, dst-endpoint, dst-port, transport-type]. For example, in a network (10.1.1.0/24),

$$\vec{E_1} = [D_1(10.1.1.1), *, D_2(10.1.1.2), 22, TCP]$$

Edge E_1 allows device D_1 to communicate with device D_2 on TCP port 22. The Edges also support URI as endpoints. For example,

$$\vec{E_2} = [D_3(10.1.1.3), *, D_4(palindrome.io), 80, TCP]$$

Edge E_2 allows device D_3 to communicate with *palindrome.io* domain endpoint on TCP port 80. The network controller resolves and maintains the mapping of domain and IP addresses. An Edge can be private, when both endpoints are internal in a network, or public when one endpoint is external, outside the network.

MUD	HANZO	Description
mud-version	No	MUD specification version
mud-url	No	Identifies MUD file
last-update	Yes	Date-and-Time of the file
is-supported	No	Device supported ?
mfg-name	Yes	Manufacturer of device
model-name	Yes	Device Model
from-device-policy	Yes	Traffic "from" the device
to-device-policy	Yes	Traffic "to" the device

TABLE I: Device profile parameters

D. Profile Model

The HANZO profile model follows the standards-based MUD profile model [5]. Table I shows the MUD profile parameters that can be derived from traffic observation. The profile uses YANG data modeling language to provide accurate and adequate usage models for network devices. The HANZO profile includes device system information metadata in *ieft-mud* root container, and communication endpoints ACLs in *ietf-access-control-list* container, as shown in Listing 1.

```
{
  "ietf-mud:mud": {
    "last-update": "2018-07-15",
    "mfg-name": "XYZ",
    "model-name": "Smart-Switch",
    "from-device-policy": {
        "access-lists": {
            "access-list": [
                {"name": "mud-01-fr"}
            ]
        }},
    "to-device-policy": {
        "access-lists": {
            "access-list": [
                {"name": "mud-01-to"}
            ]
        }}},
  "ietf-access-control-list:acls": {
    "acl": [
        {"name": "mud-01-fr", ...},
        {"name": "mud-01-to", ...}
    ]}
}
```

Listing 1: HANZO Profile with Root Containers

Network	Manufacturer	Type	Quantity
Electric	LoneyShow	Plug	4
	VOCOLinc	Plug	2
	UPSTONE	Power Strip	1
Light	TP-Link	Bulb	2
	Sengled	Bulb	2
	UPSTONE	Bulb	1
Presence	EZVIZ	Camera	1
	Wansview	Camera	1
	iHome	Motion	1
HVAC	iHome	Temperature	1
	Honeywell	Fan	1
Data	ASUS	Access Point	1
	Apple	Phone	2
	Samsung	Phone	1
	Google	Streamer	1
	Amazon	Speaker	1
	Apple	Laptop	1
	Lenovo	Laptop	1
	Lexmark	Printer	1

TABLE II: Connected-home testbed devices

IV. System Architecture

The HANZO controller architecture described in figure 4 works in three stages - extraction, generation and delegation. The decoupling allows the controller to be distributed with each stage working as an independent system. The extraction stage performs packet captures and preprocesses packet header metadata, which is passed to the generation stage. The generation stage creates a dynamic HANZO profile for each device using packet information. Finally, the delegation stage takes the generated profile containing ACLs and converts them into corresponding flow-table entries that can be installed on a SDN controller.

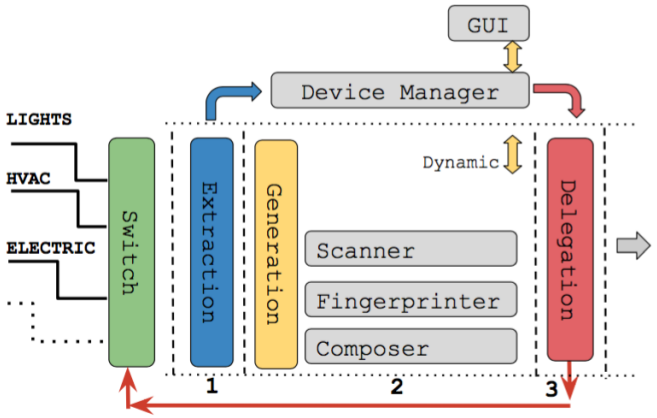

Fig. 4: HANZO controller system architecture

The Device Manager (DM) provides life-cycle management functions by maintaining device registration, configuration, operation, maintenance and quarantine states. The DM maintains the network as a directed property graph based upon the system model described in the section III. It uses *device*, *edge* and *profile* as base objects. Each device is associated with a profile object and a set of edge object. A new device in the network triggers a new instance of device object and an associated profile object. The profile object captures the generation stage output. Each new device connection results in a new edge object. The controller uses an asynchronous event processing model with all components publishing network events on a publish-subscribe event bus.

For system evaluation, the IoT testbed is modeled as a connected-home, with all devices having Wi-Fi connectivity to capture realistic general purpose daily activities. The various network devices are shown in Table II.

A. Extraction

The extraction stage captures raw traffic. It uses libpcap [17] packet capture library with pcap-filter support. Each packet is mapped as a network event following the chain of capture, parse and extract functions. The network event is represented as, { *timestamp, src-mac, dst-mac, src-net, dst-net, src-port, dst-port, trans-protocol, app-protocol, metadata* } object. The extraction stage also allows raw packet access, if subscribed.

B. Generation

The generation stage has three major modules - scanner, fingerprinter and composer. Each module maintains its own device to network events mapping. The scanner module scans for insecure device configurations such as default passwords, open ports, protocol security configurations. The fingerprinter module generates a unique device fingerprint for each device in the network by using device metadata. The composer module generates communication endpoints for each device. We further explain each module in detail.

1) Scanner: The scanner module provides vulnerability scanning function in the network. It scans a device for open ports, insecure protocol configurations, and default password interfaces by integrating the Nmap [18] security scanner. It also leverages the Nmap Scripting Engine (NSE) to provide scripts for security operations such as enumerating insecure Telnet, File Transfer Protocol (FTP), Secure Shell (SSH), Transport Layer Security (TLS) protocol configurations, default passwords, HTTP form fuzzer, and exploit kits.

2) Fingerprinter: The fingerprinter module derives device information such as manufacturer, model, type, firmware name, version, and communication pattern from extraction stage packet data. The device information is essential for correlating the manufacturer communication endpoints with the device behavior in the network. The module is further divided into two submodules - metadata and signature. The metadata module derives the product-vendor, product-type (printer, camera, sensor, speakers, etc.), firmware information using traffic heuristics methods. The signature module derives network communication patterns for each device using machine learning methods.

a) Metadata - The layer-2 MAC address can be used to identify the device manufacturer. The first 3 bytes represent Organizationally Unique Identifier (OUI) assigned by the IEEE Registration Authority [19]. The metadata module uses a local Wireshark OUI database [20] to derive device manufacturer information. For instance, the first three bytes 00:30:65 / 00:26:bb / 00:26:b0 indicate the device vendor is Apple. However, OUI lookup is not always reliable, as the network card vendor can differ from the actual device manufacturer.

The device-specific information can also be derived from application layer protocol such as DHCP [21]. The DHCP Options - Option 55 (Parameter Request List), DHCP Option 60 (Vendor Class Identifier), and Option 43 (Vendor Specific Information) provide vendor and device related information. For example, the Fingerbank service [22] provides device fingerprint information such as operating system, manufacturer, OS type using a collaborative platform to share device DHCP fingerprints. Furthermore, the vendor information can also be derived from DNS requests as most devices communicate with cloud setup services during initialization. The metadata module monitors DHCP and DNS requests for deriving this information. It also performs a WHOIS lookup [23] of observed device IP addresses, which can provide additional manufacturer information.

A single method cannot derive all the device-specific information, therefore the module uses the combination of all the above methods to derive device metadata. Table III shows metadata module output for each device. For manufacturer name, the WHOIS method takes preference than OUI method.

b) Signature - The signature module differentiates IoT devices from general-purpose devices in the network using network traffic classification. It further derives IoT device types such as printer, camera, sensor, speakers, etc. using a traffic behavior profile of each device.

The module extracts the traffic signature of each device by analyzing protocol headers of each networking layer - link, network, transport, and application. Each signature signals different class of device in the network, and helps correlate network traffic patterns of similar classes of devices. For example, a printer with multiple large usage packets will have a different signature from a light bulb, with relatively small usage packets.

Vendor	OUI	WHOIS	OS	Thing
EZVIZ	AmpakTec	Nexperian	Linux	Yes
WANSView	Shenzen	Null	Linux	Yes
iHome	Azurewav	Evrythng	Java ME	Yes
LoneyShow	Espressi	Hangzhou	Null	Yes
Tp-Link	TP-LINK	TP-LINK	Null	Yes
Sengled	Expressi	Sengled	Null	Yes
Apple	Apple	Apple	iOS	No
Samsung	Samsung	Samsung	Android	No

TABLE III: Device Metadata and Signature

The module takes raw packets from the extraction stage as inputs and creates a feature vector with transaction, session and flow attributes [24]. The vector is passed to the device classifier, which uses an ensemble learning classifier [25] with Decision Tree [26], Support Vector Machines (SVM) [27] and k-Nearest Neighbors (k-NN) [28] methods as base estimators. The module works in two phases, in the first phase, it classifies IoT and general-purpose devices. In the second phase, it further classifies IoT devices into specific device types. The second-phase classifier is currently trained for five device types - printer, camera, light bulb, powerplug and speakers.

3) Composer: The composer module maps a set of valid public communication endpoints for each device during traffic observation window having 5-tuple as explained in section III. On device network initialization, it generally communicates with its manufacturer services, such as device management, user setup, storage, and application cloud services specific to its function. Typically, these endpoints are established in the initial hours of device operation, and do not change when compared to general-purpose devices. Using heuristics, the composer module derives all public endpoints of a device by traffic observation. Table IV shows that a observation time window of 10 minutes is sufficient for almost all devices to derive public communication endpoints.

Manufacturer/Device	1 M	5 M	10 M	20 M	30 M
EZVIZ/Camera	3	3	4	4	4
WANSView/Camera	3	4	10	10	10
iHome/Monitor	4	4	4	4	4
UPSTONE/Plug	4	4	4	4	4
TP-LINK/Bulb	2	2	2	2	2
Sengled/Bulb	2	2	2	2	2

TABLE IV: Public endpoints with observation window

The DM compiles a final HANZO profile using the generative stage modules output in MUD profile format. Listing 2 shows a generated endpoint rule.

C. Delegation

The delegation stage takes the generated profile ACLs and converts them into SDN flow-table rules. It delegates the ACL enforcement to a SDN controller that implements flow-table rules on a SDN switch.

```
{
  "name": "mud-01-fr",
  "type": "ipv4-acl-type",
  "matches": {
    "ipv4": {
      "ietf-acldns:src-dnsname":
            "www.xyz-cloud.com"
    },
    "source-port-range": {
      "lower-port": 80,
      "upper-port": 80
    },
    "tcp": {
      "ietf-mud:direction-initiated":
            "from-device"
    }
  },
  "actions": {
    "forwarding": "accept"
  }
}
```

Listing 2: Generated Device Rule Example

D. HANZO and MUD

As discussed in the previous section, the *dynamically* generated HANZO profile uses the MUD profile format. For any future MUD-enabled IoT devices, the HANZO controller also supports the device manufacturer's *statically* generated MUD profiles. The controller support for both profile types creates a backward compatible unified network management platform.

The controller provides MUD support by using either of the DHCP Options headers, OPTION_MUD_URL_V4 (161) for IPv4 or OPTION_MUD_URL_V6 (112) [5] for IPv6 respectively. For devices that support MUD-URLs, the extraction stage captures the string and publishes a MUD network event. The DM captures the event and downloads the MUD profile. For devices that don't support MUD-URLs, the controller follows the regular HANZO profile generation process. As both profiles share the same format, the resulting ACLs are implemented similarily in the network.

Furthermore, for MUD-enabled devices, in addition to downloading the MUD profile, the controller also generates the HANZO profile for the same device. The DM compares the static MUD profile endpoints with the dynamic HANZO profile endpoints. If a difference is detected in the endpoints, the DM publishes a security anomaly event for the compromised device along with the observed device endpoints. The DM generates and compares HANZO profiles periodically to provide continous security monitoring.

V. Implementation

The HANZO controller prototype is implemented as a network gateway with all components running on a single

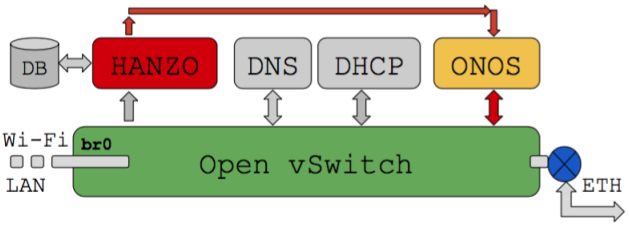

Fig. 5: HANZO controller implementation

node. The node is running Ubuntu Linux 16.04 on an Intel NUC hardware with 8 GB RAM, Gigabit Ethernet, Wi-Fi 802.11ac, 256 GB storage. The ethernet interface is configured as external internet link, and Wi-Fi interface is configured as an access point (AP) for local devices. The NUC is running a local DHCP server on UDP port 67, and DNS server on UDP port 53 listening on the AP interface. The Open vSwitch (OvS) [29] is used as an OpenFlow switch. The data plane enforcement is done via Open Network Operating System (ONOS) SDN controller [30]. The OvS bridge interface (br0) is configured with Wi-Fi interface port as shown in figure 5.

The network monitoring and management is performed via a web application GUI service. The web application provides network topology visualisation and basic management functions including Create/Read/Update/Delete (CRUD) for devices and configuration debugging. All controller stages are implemented using Python. The extraction stage is using the Scapy [31] framework. The generation stage is using python-nmap, scikit-learn [32] frameworks. The delegation stage is using python-json and python-http modules with ONOS controller API [33]. All data is stored in a local MongoDB [34] database.

Device	IoT	Profile	Endpoints
LoneyShow/Plug	Yes	Yes	2
VOCOLinc/Plug	Yes	Yes	2
UPSTONE/Power Strip	Yes	Yes	4
TP-Link/Bulb	Yes	Yes	2
Sengled/Bulb	Yes	Yes	2
UPSTONE/Bulb	Yes	Yes	6
EZVIZ/Camera	Yes	Yes	6
Wansview/Camera	Yes	Yes	10
iHome/Motion	Yes	Yes	4
iHome/Temperature	Yes	Yes	4
Honeywell/Fan	Yes	Yes	4
ASUS/AP	No	No	-
Apple/Phone	No	No	-
Samsung/Phone	No	No	-
Google/Streamer	No	No	-
Amazon/Speaker	Yes	Yes	18
Apple/Laptop	No	No	-
Lenovo/Laptop	No	No	-
Lexmark/Printer	Yes	Yes	4

TABLE V: HANZO device evaluation

VI. Evaluation

The HANZO controller was evaluated against the connected home testbed with both IoT and general-purpose devices described in section IV. Table V shows the controller correctly differentiates between IoT and general-purpose devices with 100% accuracy. The controller generates device profile with communication endpoints after the ideal 10 mins window derived from previous observations. The monitoring experiment was repeated 20 times for each device with similar results observed everytime.

The device metadata module that derives manufacturer name, device type, device model using heuristics was not consistent in capturing all the required information. For devices, in most instances, the network interface card, e.g., Espressi, was found to be different from device manufacturer showing diverse vendor supply-chain. The DHCP device information methods were useful for only general-purpose devices. Except Lexmark printer, all evaluated device manufacturers do not use DHCP headers for sharing device information with the network. Although, the first contact DNS request was consistent everytime for each device, some devices contacted public NTP servers first before contacting device management servers.

VII. Related Work

In Mirai worm architecture analysis, the authors [1] discuss the need for baseline security hardening, automatic updates, attack notification mechanisms, facilitating device identification, and proper device life-cycle management. The HANZO architecture tries to address these concerns with a systematic approach to IoT security.

The IoT Sentinel architecture [35] describes a device-type (make, model, software version) identification method using machine learning classification that triggers automatic security policy enforcement levels - strict, restricted and trusted, for vulnerable devices. The project proposes a hybrid deployment model of two major components - a local security gateway and a cloud security service to augment local functions of device identification and vulnerability assessment. In contrast, HANZO architecture does not include any cloud-based processing component. All system stages are processed locally with only exception of external device metadata API access such as fingerbank and WHOIS records. Furthermore, the Sentinel architecture does not differentiate between IoT and general-purpose devices, and the generated profile format is not standards based.

The DIoT project [36] improves the Sentinel architecture by also learning device communication behavior patterns to generate access control policies. The Kalis [37] project presents a similar self-adapting learning Intrusion Detection System for IoT networks. Both these frameworks employ device operational behavior for anomaly detection that is used for device isolation. In contrast, our architecture supports anomaly detection using continous device profile comparison method.

The IoT Inspector [38] project monitors device traffic in a real smart home network setup for privacy and security issues. This includes identifying communication endpoints the home devices talk to with an emphasis on the type of data collected. The IoTSec proposal [39] highlights the need for a network-first approach where the local network plays a critical role in securing its boundaries. The authors argue rethinking network security in three key dimensions: security policies abstraction, dynamic access-control policies, and context-aware enforcement capabilities. Our proposed network-centric architecture tries to address the points higlighted by this proposal.

VIII. Conclusion and Future Work

We present a network management architecture to secure IoT devices by limiting the communication endpoints. We model a network with device, edge and profile entities for configuration and operations management. We describe the HANZO controller, a modular system with extraction, generation and delegation stages resulting in a device profile that is enforced in the network using an SDN controller. We show that all required parameters of a MUD profile can be derived from traffic observation. We evaluate the system using a connected-home testbed with both IoT devices and general-purpose devices.

For our future work, we are integrating a traffic behavior module for anomaly detection in the device operational state. Currently, the generated device profile is enforced at the network gateway. We are exploring multi-level enforcement models for large-scale networks. A transparent public crowd-sourced knowledge base that evaluates and rates manufacturers based upon device operations will create a healthy IoT ecosystem. We are also exploring the use of anonymized device configuration data to create a public knowledge base.

Acknowledgment

The authors would like to acknowledge Xuan He, Akhilesh Srivastava, Alexander Bienstock, Adithya Beemanapalli and Xuxiang Wu for contributions to the initial prototype implementation. We would also like to thank Eliot Lear for discussing the MUD specification.

References

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

[2] B. Krebs, "IoT Devices as Proxies for Cybercrime," 2016. [Online]. Available: https://krebsonsecurity.com/2016/10/iot-devices-as-proxies-for-cybercrime

[3] PenTestPartners, "Thermostat Ransomeware: a Lesson in IoT Security," 2016. [Online]. Available: https://www.pentestpartners.com/security-blog/thermostat-ransomware-a-lesson-in-iot-security

[4] Symantec Corporation, "Internet Security Threat Report," 2018. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf

[5] R. D. E. Lear and D. Romascanu, "Manufacturer Usage Description Specification," IETF Draft, June 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-opsawg-mud-25

[6] M. Bjorklund, "The YANG 1.1 Data Modeling Language," RFC 7950, August 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7950.txt

[7] M. Jethanandani, L. Huang, S. Agarwal, and D. Blair, "Network Access Control List (ACL) YANG Data Model," IETF Draft, October 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-netmod-acl-model-20

[8] M. J. Covington and R. Carskadden, "Threat Implications of the Internet of Things," in *The 5th International Conference on Cyber Conflict (CyCon)*, Tallinn, Estonia, June 2013, pp. 1–12.

[9] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins, "Evaluation of the Ability of the Shodan Search Engine to Identify Internet-facing Industrial Control Devices," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 2, pp. 114–123, 2014.

[10] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine backed by Internet-wide Scanning," in *The 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, Colorado, USA, October 2015, pp. 542–553.

[11] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the Rise of IoT Compromises," in *The 9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., USA, August 2015.

[12] S. Edwards and I. Profetis, "Hajime: Analysis of a decentralized internet worm for iot devices," *Rapidity Networks*, vol. 16, 2016.

[13] D. Goodin, "Brickerbot, the permanent denial-of-service botnet, is back with a vengeance," *Ars Technica*, 2017.

[14] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748, June 2004. [Online]. Available: https://tools.ietf.org/html/rfc3748

[15] R. Droms and W. Arbaugh, "Authentication for DHCP Messages," RFC 3118, June 2001. [Online]. Available: https://tools.ietf.org/html/rfc3118

[16] L. Seitz, G. Selander, and C. Gehrmann, "Authorization framework for the internet-of-things," in *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, June 2013, pp. 1–6.

[17] TCPDUMP Project, "Libpcap: Network Traffic Capture Library," 2018. [Online]. Available: http://www.tcpdump.org/

[18] Nmap Project, "Nmap Security Scanner," 2018. [Online]. Available: https://nmap.org/

[19] IEEE Registration Authority, "Ogranizationally Unique Identifier (OUI)," 2018. [Online]. Available: https://standards.ieee.org/develop/regauth/oui/index.html

[20] Wireshark Project, "Wireshark Manufacturer Database," 2018. [Online]. Available: https://www.wireshark.org/tools/oui-lookup.html

[21] S. Alexander and R. Droms, "DHCP Options and BOOTP Vendor Extensions," RFC 2132, March 1997. [Online]. Available: https://tools.ietf.org/html/rfc2132

[22] Inverse Inc., "Fingerbank API," 2018. [Online]. Available: https://fingerbank.org

[23] L. Daigle, "Whois protocol specification," *RFC 3912*, 2004.

[24] D. Bekerman, B. Shapira, L. Rokach, and A. Bar, "Unknown malware detection using network traffic classification," in *IEEE Conference on Communications and Network Security (CNS)*, Florence, Italy, September 2015, pp. 134–142.

[25] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, Cagliari, Italy, June 2000, pp. 1–15.

[26] S. R. Safavian and D. Landgrebe, "A Survey of Decision Tree Classifier Methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[27] T. Joachims, "Making large-scale svm learning practical," Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund, Tech. Rep., 1998.

[28] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance Metric Learning for Large Margin Nearest Neighbor Classification," in *Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, December 2006, pp. 1473–1480.

[29] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch." in *The 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Oakland, CA, USA, May 2015, pp. 117–130.

[30] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, Chicago, IL, USA, August 2014, pp. 1–6.

[31] Scapy Project, "Packet Crafting for Python," 2018. [Online]. Available: https://scapy.net/

[32] SciKit-Learn Project, "Machine Learning in Python," 2018. [Online]. Available: http://scikit-learn.org/

[33] ONOS Project, "Open Network Operating System," 2018. [Online]. Available: https://wiki.onosproject.org/display/ONOS/REST

[34] MongoDB Project, "Mongo Database," 2018. [Online]. Available: https://github.com/mongodb/mongo

[35] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma, "IoT Sentinel: Automated Device-Type Identification for Security Enforcement in IoT," in *The 37th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Atlanta, GA, USA, June 2016, pp. 2177–2184.

[36] T. D. Nguyen, S. Marchal, M. Miettinen, M. H. Dang, N. Asokan, and A. Sadeghi, "Dïot: A crowdsourced self-learning approach for detecting compromised iot devices," *CoRR*, vol. abs/1804.07474, 2018. [Online]. Available: http://arxiv.org/abs/1804.07474

[37] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis — A System for Knowledge-Driven Adaptable Intrusion Detection for the Internet of Things," in *The 37th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Atlanta, GA, USA, June 2017, pp. 656–666.

[38] G. Chu, N. Apthorpe, and N. Feamster, "Security and privacy analyses of internet of things toys," *CoRR*, vol. abs/1805.02751, 2018. [Online]. Available: http://arxiv.org/abs/1805.02751

[39] 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 *The 14th ACM Workshop on Hot Topics in Networks*, Philadelphia, PA, USA, November 2015.