IoT Device Profiling: From MUD Files to S×C Contracts

Guðni Matthíasson [1], Alberto Giaretta [2], Nicola Dragoni [3]

Abstract: Security is a serious, and often neglected, issue in the Internet of Things (IoT). In order to improve IoT security, researchers proposed to use Security-by-Contract (S×C), a paradigm originally designed for mobile application platforms. However, S×C assumes that manufacturers equip their devices with security contracts, which makes hard to integrate legacy devices with S×C. In this paper, we explore a method to extract S×C contracts from legacy devices' Manufacturer Usage Descriptions (MUDs). We tested our solution on 28 different MUD files, and we show that it is possible to create basic S×C contracts, paving the way to complete extraction tools.

Keywords: Internet of Things; S×C; Security-by-Contract; MUD; Manufacturer Usage Description; Device profiling

1 Introduction

The Internet of Things (IoT) is becoming more and more pervasive in our society. With the increasing number of connected devices come additional security risks. IoT devices are usually simple and resource constrained, which also means that they tend to have a limited capacity for security routines. Moreover, in order to gain market shares, manufacturers tend to prioritise easily perceivable features over security [DGM18]. As a result, hackers have been targeting IoT devices for various purposes: distributed denial-of-service (DDoS) attacks [Go], cryptocurrency mining [An], espionage, and many others [Hi]. These types of attacks are expected to become more common as the IoT expands.

Security-by-Contract (S×C) is a promising paradigm for mitigating some IoT security issues. Originally proposed for mobile applications [Dr07], S×C envisions devices that carry security contracts, easy to validate and verify against network security policies [GDM19b]. The S×C framework utilises the fog computing paradigm, which extends the concept of cloud computing by adding a middle layer between the end devices and the cloud. Practically, this middle layer consists of fog nodes, machines dedicated to data aggregation and data processing. Fog nodes are distributed and localised, allowing lower latency for time-sensitive tasks with respect to the cloud. They also provide a more local and controlled storage

[1] Technical University of Denmark (DTU), Department of Applied Mathematics and Computer Science, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark s190064@student.dtu.dk
[2] Örebro University, AASS Research Centre, Fakultetsgatan 1, 702 81 Örebro, Sweden alberto.giaretta@oru.se
[3] Technical University of Denmark (DTU), Department of Applied Mathematics and Computer Science, Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark and Örebro University, AASS Research Centre, Fakultetsgatan 1, 702 81 Örebro, Sweden ndra@dtu.dk

location for sensitive data. In the fog computing paradigm, the cloud can still be used for less time-sensitive and security-sensitive operations. These features make fog computing especially useful for IoT networks where latency and data security are important. They also make fog nodes ideal for security-critical roles within a network.

When a new device first connects, it provides a fog node with a contract formally describing its intended behaviour on the network, denoted as a list of security rules. An example of such a rule for a Philips Hue White smart lighting system can be seen in Tab. 1. This rule allows other Philips devices to access its *On*, *Bri* and *Hue* services, and requires access to the HueMotion *Presence* service over the local area network (LAN).

	Rule R_{B1}
D	PHILIPS.HUEWHITE
DOM	LAN
SHARES	PHILIPS.*
PROVIDES	ON, BRI, HUE
REQUIRES	PHILIPS.HUEMOTION.PRESENCE

Tab. 1: A security rule for the Philips Hue White smart lighting system [GDM19a]

Similarly, a security policy is a set of rules which describes the behaviours allowed within the network. In S×C, fog nodes act as security gateways on the local network. They are responsible for verifying and validating contracts, as well as for maintaining and enforcing the security policy. Upon receiving a device contract, the fog node validates it against the existing security policy. The validation phase checks the contract for inconsistencies and tests if the rules violate the existing security policy; if the contract is valid, the fog node adds the rules to the policy. This helps to ensure an up-to-date, specific, and internally consistent security policy, providing a good basis for identifying abnormal behaviour on the network.

Contribution of the Paper In an ideal world, manufacturers would produce contracts and store them in their devices, and this is not a realistic short-term goal. We have to face thousands of devices on the market which cannot naturally comply with S×C. But a growing number of these devices are compliant with the Manufacturer Usage Description (MUD) specification [LDR19], an Internet Engineering Task Force (IETF) standard which allows devices to signal to the network their requirements in order to work properly. As shown in Fig. 1, we propose a method for integrating MUD-compliant devices with an S×C framework. Our approach is based on extracting S×C contracts from MUD definitions, by means of access control list (ACL) analysis, Dynamic Host Configuration Protocol (DHCP) fingerprints and queries to the Fingerbank application programming interface (API).

For S×C to achieve widespread use, we need a way for analysing a device behaviour, before we can generate a suitable contract and grant network access. The S×C framework shows great promise in terms of sustainable security on a local network, but in its current state it

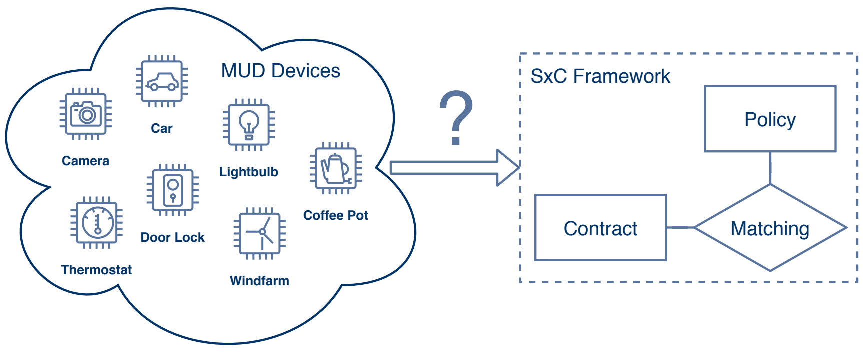

Fig. 1: MUD-compliant devices are growing in number. How can we integrate such devices with an S×C framework? We need a method for extracting S×C contracts from MUD definitions.

requires drastic additions to the development and manufacturing processes of IoT devices. Through an experiment performed on 28 different MUD files, we show that it is possible to extract basic S×C contracts from MUD ACL specifications. Even though the resulting contracts are partial, our work paves a promising way for profiling MUD devices and extracting complete S×C contracts.

Paper Outline The paper is organised as follows. In Sect. 2 we give an overview of related work. In Sect. 3 we present our proposal for extracting S×C contracts from MUD files, and in Sect. 4 we evaluate our results. Limitations and next steps to achieve complete S×C contracts are discussed in Sect. 5. Last, in Sect. 6 we draw our conclusions.

2 Related Work

MUD is a standard meant to allow devices to describe their requirements in order to function properly. A MUD file describes the types of communication a device establishes under normal operating conditions. Namely, it provides access control lists for both inbound and outbound communication, grouped by protocols. According to the standard, a MUD file is hosted on servers run by the device manufacturer, and the device stores a link to its MUD file as a DHCP option [LDR19].

MUD strives to achieve similar goals as S×C, but it does not go as far in describing device-based communication patterns. It also differs from S×C in that the MUD file is not provided directly by the device, but rather by an online server. Thus, an internet connection is required for MUD to function. The research community anticipated issues similar to those described in Sect. 1 and Hamza et al. [Ha18] came up with a way of generating MUD files for devices from behavioural analysis. In their work, they collected packets for 28 IoT devices over 6 months and produced the related proof-of-concept MUD files [Ha].

DHCP is a protocol for dynamically assigning IP addresses to network devices. The protocol specifies several parameters for the initial *DHCP DISCOVER* packet including option 55, the Parameter Request List, which allows a device to request configuration information from the DHCP server [Al].The specific information fields requested, and the order in which they are listed, are usually manufacturer- and often device-specific. To the degree that it is commonly referred to as a *DHCP fingerprint*. Fingerbank is an online database that collects DHCP fingerprints and pairs them with device profiles which contain useful information, such as the device name and manufacturer. It offers an API that allows a user to query the database with DHCP DISCOVER packet data and replies with the relevant device information [Fi].

Thomsen [Th19] proposed a method of determining the device type (lamp, speaker, etc.) using a Random Forest Machine-Learning algorithm. His research included extracting information from MUD files for the purpose of this classification. Thomsen's approach to MUD-based device type classification revolved around the MUD file *systeminfo* field. This identifier was used as a query string for an online search. The results of the search were scraped for text, which was then fed to the classification algorithm. The *systeminfo* field is the only information from the MUD files utilised in the classification process. This paper investigates what other relevant information can be extracted from the MUD files and to what degree contracts can be generated based on that information.

Several papers have been published on the topic of profiling IoT device behaviour on a network outside the context of S×C. Notable examples include IoT SENTINEL by Miettinen et al. [Mi17], IoTSense by Bezawada et al. [Be18] and AuDI (Autonomous IoT Device-Type-Identification) by Marchal et al. [Sa19]. All of these solutions include allowing a new device to connect and passively observing its behaviour after the fact. This approach may be required to assemble a complete profile for an unknown device but it does represent a compromise in the pursuit of preemptive profiling. The insights provided by the aforementioned research are not discussed in individual detail in this paper, but instead recommended as potentially useful methods for further progress along this line of research.

3 From MUD Files to S×C Contracts

As aforementioned, in this paper we decided to focus on MUD files. The choice was driven by three main reasons:

1. MUD is designed for different environments, from home networks to larger ones. It is reasonable to assume that MUD will become widely adopted in the future.
2. Hamza et al.[Ha18] showed a reliable method for generating MUD profiles for non-MUD devices.

3. Even though MUD does not provide enough information to build a full security contract, it gives helpful information about the devices’ expected communications.

For the purposes of this research, the main point of interest in MUD files are their ACL specifications. ACLs provide information on communication patterns, including domain names (in the case of Internet communication), as well as ports and protocols involved. The first question is: are MUD ACLs enough to create an S×C contract? The information required for building complete S×C contracts is:

1. Device name and manufacturer
2. Domain of communication (LAN/internet)
3. List of devices which the device can communicate with
4. List of services provided and required
5. ACL in terms of identified devices and services

Manufacturer and device names are not stored in the MUD file but they can be retrieved by feeding DHCP fingerprints to the Fingerbank API. Also, MUD ACLs provide a list of source and destination ports used for LAN and internet communications. However, with MUD devices we miss the services required and provided, as well as a list of other devices which the device can communicate with. MUD is not sufficient to construct complete security contracts for S×C, but it provides useful information. The goal of our work is to see how much useful data we can extract from the 28 MUD ACLs provided by Hamza et al. [Ha18].

3.1 Implementation

The first step was to create a fork of Thomsen’s codebase [Th19]. Additional code was then written to break up and extract information from the MUD ACLs [Ma].We determined that the following information could be reliably extracted: ports used for communication, grouped by LAN/internet and local/remote, and domains communicating with the device over the internet, grouped by inbound/outbound.

The profiling solution, depicted in Fig. 2, consists of the following steps:

1. **Receive DHCP DISCOVER request containing MUD URL**
2. **Extract DHCP fingerprint, user-agent string, media access control (MAC) address and MUD URL** This info is included in the DHCP DISCOVER packet.
3. **Query the Fingerbank API for manufacturer and device names using the extracted client information** The API accepts a DHCP fingerprint, user-agent string and MAC address.

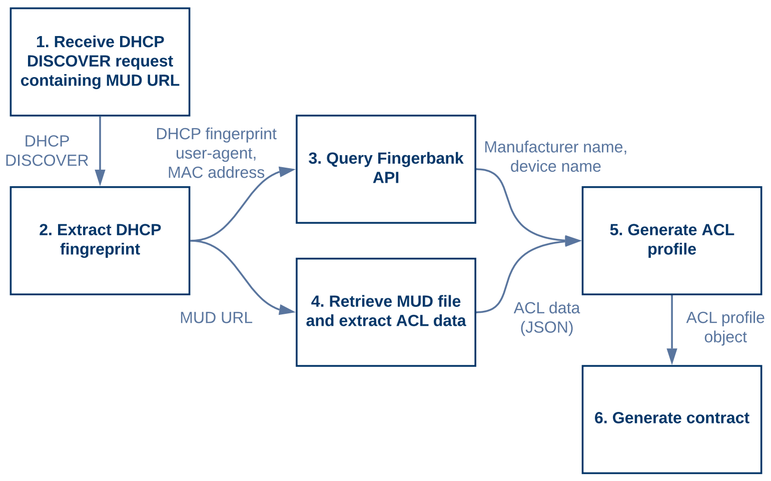
1. Receive DHCP DISCOVER request containing MUD URL
DHCP DISCOVER
DHCP fingerprint user-agent, MAC address
3. Query Fingerbank API
Manufacturer name, device name
2. Extract DHCP fingreprint
5. Generate ACL profile
MUD URL
4. Retrieve MUD file and extract ACL data
ACL data (JSON)
ACL profile object
6. Generate contract

Fig. 2: This workflow shows our proposed approach for extracting S×C contracts from MUD ACLs.

4. **Retrieve MUD file from extracted URL and extract the ACLs** We made some base assumptions about the ACL data: 1) each local port represents a provided service and 2) each remote port represents a required service.
5. **Generate an ACL profile object using manufacturer name, device name, and ACLs** We defined a data model class, the ACL profile, to encapsulate relevant ACL information, along with functionality to instantiate such objects from raw ACL data. The protocols used are also available for extraction, but it is unclear how they would help in constructing contracts for S×C, so they were not included in this model.
6. **Generate a contract object using the ACL profile object** We defined a second data model class, the contract, to represent a security contract, complete with functionality to instantiate contracts from ACL profile objects.

4 Evaluation

For the purpose of testing this functionality, we decided to use unique identifiers, acquired by manually looking up each device, for Fingerbank API lookups. This was necessary due to the lack of physical devices to test and scarcity of raw DHCP fingerprint examples readily available online. Because of this, we were unable to determine the reliability of identifying device and manufacturer names with DHCP fingerprint lookups. We wrote a script to run the modified MUD profiling solution on each of the 28 MUD files provided.

```
Device type:            Camera
Classification score:   0.17817759870529015
Name:                   Belkin.NetCam
Mud file ACLs:
Security contract Belkin.NetCam:
        Rule Belkin.NetCam.all:
                Device:     Belkin.NetCam
                Domain:     *
                Shares:     *
                Provides:   {'5104'}
                Requires:   {'5104'}
        Rule Belkin.NetCam.lan:
                Device:     Belkin.NetCam
                Domain:     LAN
                Shares:     *
                Provides:   {'67', '3478', '53'}
                Requires:   {'67', '3478', '1900', '53'}
        Rule Belkin.NetCam.net:
                Device:     Belkin.NetCam
                Domain:     Internet
                Shares:     *
                Provides:   ['443', '8443', '8899', '123', '3475']
                Requires:   ['443', '8443', '8899', '123', '3475']

Contact domains:
        nat.xbcs.net
        api.xbcs.net
        [..]
```

Fig. 3: Test output for a Belkin Camera device

Fig. 3 shows a test output example for a Belkin camera device. The generated partial contract represents the following communication profile:

- The camera communicates on port 5104 over both LAN and the internet.
- The camera communicates on ports 53, 67 and 3478, and transmits to remote port 1900 over LAN.
- The camera communicates on ports 8899, 123, 3475, 8443 and 443 over the internet.

The raw test output can be found on the GitHub repository [Ma].As stated in Sect. 3, the basic ACL data also contains information on the domains contacted over the internet, as well as the protocols used. We excluded the protocols from the basic contract model for simplicity. Sect. A presents the results in a condensed format where the domain names identified are omitted. Instead, the number of domain names extracted from the MUD ACLs is specified along with the list of identified ports used by each device for inbound and outbound communication, on the local network and the internet.

This data may be used to define enforceable security policies on a network, by putting restrictions on which external domains can contact a device, using which ports and protocols, and which ports can be used for local network communication. However, this is not enough for describing devices' behaviour to the degree required by S×C contracts.

5 Future Work

The contracts we produced with our method can be used to achieve a basic behavioural whitelist. But they do not encompass the entire behavioural profile of a device. There is a question left: how can we improve our output S×C contracts? Based on our results, we suggest two potential approaches:

1. Define a general contract for each device type, select for every new device the appropriate contract based on the type classification, and use MUD ACL data to narrow it down.
2. Add an intermediate, *tentative* state to the S×C process. A new device is granted access to the network, limited by the MUD ACL, while additional profiling takes place and a valid contract is generated.

The first option would define which external domains the device could communicate under standard conditions. This could provide the S×C fog node with a baseline for identifying abnormal communications, but it would not be perfect. For example, this approach might produce contracts too general, granting unnecessary permissions. The second option would produce better contracts, as the data described in Sect. 4 would be enhanced with observed network data, allowing device-specific and service-specific permissions. But it would also require the S×C fog node to actively monitor new devices communications while they are in this *tentative* access state, increasing the computational burden.

Both options could also be combined, whereby a temporary contract could be created by augmenting an existing general type-specific contract for the analysis period. During this period, a more specific contract could be generated based on a more thorough and sophisticated behaviour analysis. Methods for such analysis are rapidly emerging, as mentioned with some notable examples in Sect. 2.

6 Conclusion

The market demand for IoT devices has considerably outpaced the development of secure IoT solutions. Security-by-Contract (S×C) attempts to improve the IoT shortcomings, with respect to security configurability and lacking behavioural descriptions. At the time of writing, one issue S×C has to face is its compatibility with existing technology. In this paper, we presented Manufacturer Usage Description (MUD), an IETF standard which describes basic requirements for compliant IoT devices. Within MUD files, we identified information for extracting S×C contracts, and integrating MUD-compliant devices in S×C frameworks.

Then, we proposed a method to extract such information and, in order to verify our hypothesis, we applied this method to 28 different MUD files. Our experiment shows that it is possible, indeed, to extract some useful information for basic S×C contracts. However, we show that our method outputs only partial S×C contracts. We have also identified two potential methods for extracting valid and useful S×C contracts for previously unknown devices. Both include the approximation of a valid contract from ACLs, device type, and further behaviour analysis.

With the increase in resource-constrained IoT devices on the market, we are facing an increase in attack surface. This presents a huge challenge for cybersecurity, but the growing research on IoT security is promising for the future of the Internet.

Appendix A Test results

MUD File	**S×C Device**	**LAN**		**Internet**		**# Dom.**
amazonEcho	Amazon.Echo	5353	O	33434	I/O	20
		1900	O	443	I/O	
		67	I/O	123	I/O	
		53	I/O	89	I/O	
augustdoorbellcam	August.DoorBellCamera	67	I/O	443	I/O	19
		53	I/O			
		547	I/O			
awairAirQuality	Awair.R2	67	I/O	8883	I/O	3
		53	I/O	443	I/O	
belkincamera	Belkin.NetCam	5104	I/O	8899	I/O	8
		3478	I/O	8443	I/O	
		1900	O	5104	I/O	
		67	I/O	3475	I/O	
		53	I/O	443	I/O	
				123	I/O	
blipcareBPmeter	BLIP.Systems	67	I/O	8777	I/O	1
		53	I/O			
canaryCamera	Canary.All-in-One	67	I/O	443	I/O	8
		53	I/O	80	I/O	
chromecastUltra	Google.ChromecastUltra	5353	O	5228	I/O	37
		1900	O	443	I/O	
		67	I/O	123	I/O	
		53	I/O	80	I/O	
dropcam	Nest.Camera	67	I/O	443	I/O	4
		53	I/O	123	I/O	
hellobarbie	Nabi.BarbieTablet	67	I/O	443	I/O	3
		53	I/O			

MUD File	S×C Device	LAN		Internet		# Dom.
hpprinter	HP.Printer	5355	O	5223	I/O	3
		5353	O	5222	I/O	
		547	I/O	443	I/O	
		137	O	80	I/O	
		67	I/O			
		53	I/O			
HueBulb	Philips.PhilipsHueSmartlighting	5353	O	443	I/O	12
		1900	O	123	I/O	
		67	I/O	80	I/O	
		53	I/O			
ihomepowerplug	iHome.SmartPlug	5353	O	443	I/O	2
		67	I/O	80	I/O	
		53	I/O			
lifxbulb	LIFX.lighting	56700	O	56700	I/O	2
		67	I/O	123	I/O	
		53	I/O			
nestsmokesensor	Nest.Smoke+COAlarm	67	I/O	11095	I/O	46
		53	I/O			
NetatmoCamera	Netatmo.Camera	67	I/O	4500	I/O	12
		53	I/O	500	I/O	
				443	I/O	
				123	I/O	
				80	I/O	
NetatmoWeatherStation	Netatmo.PersonalWeatherStations	67	I/O	25050	I/O	1
		53	I/O			
pixstarphotoframe	Pix-Star.WiFiFrame	138	O	443	I/O	2
		137	O	80	I/O	
		67	I/O			
		53	I/O			
ringdoorbell	Ring.Doorbell	67	I/O	9998	I/O	4
		53	I/O	443	I/O	
				123	I/O	
				80	I/O	
samsungsmartcam	Samsung.IPCamera	5353	O	5222	I/O	5
		1900	O	443	I/O	
		67	I/O	123	I/O	
		53	I/O			
SmartThings	Samsung.SmartThings	1900	O	443	I/O	3
		67	I/O	123	I/O	
		53	I/O			
tplinkcamera	TP-Link.IPCamera	5353	O	3478	I/O	6
		67	I/O	443	I/O	
		53	I/O	123	I/O	
				80	I/O	
tplinkplug	TP-Link.HS100	67	I/O	50443	I/O	11
		53	I/O	123	I/O	

MUD File	S×C Device	LAN		Internet		# Dom.
tribyspeaker	Invoxia.SmartPortableSpeaker	5353	O	10003	O	14
		67	I/O	10002	I/O	
		53	I/O	8090	I/O	
				5228	I/O	
				443	I/O	
				123	I/O	
				80	I/O	
wemomotion	Belkin.WeMo	1900	O	8899	I/O	3
		123	I/O	8443	I/O	
		67	I/O	3478	I/O	
		53	I/O			
wemoswitch	Belkin.SmartHome	1900	O	8443	I/O	2
		3478	I/O	3475	I/O	
		123	I/O			
		67	I/O			
		53	I/O			
withingsbabymonitor	Withings.SBM	5353	O	1935	I/O	7
		67	I/O	80	I/O	
		53	I/O			
withingscardio	Nokia.-WithingsIoT	67	I/O	443	I/O	1
		53	I/O			
withingssleepsensor	Withings.AURA	5353	O	443	I/O	1
		67	I/O	80	I/O	
		53	I/O			

References

[Al] Alexander, S.: RFC 2132, `https://tools.ietf.org/html/rfc2132`, Accessed: 2020-04-09.

[An] Anonymous: Linux Worm targets Internet-enabled Home appliances to Mine Cryptocurrencies, `https://thehackernews.com/2014/03/linux-worm-targets-internet-enabled.html`, Accessed: 2020-04-09.

[Be18] Bezawada, B. et al.: Behavioral Fingerprinting of IoT Devices. In: Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security. ASHES '18, Association for Computing Machinery, Toronto, Canada, pp. 41–50, 2018, ISBN: 9781450359962, URL: `https://doi.org/10.1145/3266444.3266452`.

[DGM18] Dragoni, N.; Giaretta, A.; Mazzara, M.: The Internet of Hackable Things. In: Proceedings of 5th International Conference in Software Engineering for Defence Applications. Springer International Publishing, Rome, Italy, pp. 129–140, 2018, ISBN: 978-3-319-70578-1.

[Dr07] Dragoni, N. et al.: A Security-by-Contract Architecture for Pervasive Services. In: Third International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SecPerU 2007). IEEE, pp. 49–54, 2007.

[Fi] Fingerbank: Device Fingerprints, https://fingerbank.org/, Accessed: 2020-04-09.

[GDM19a] Giaretta, A.; Dragoni, N.; Massacci, F.: IoT Security Configurability with Security-by-Contract. eng, Sensors (Basel, Switzerland) 19/19, Sept. 2019.

[GDM19b] Giaretta, A.; Dragoni, N.; Massacci, F.: Protecting the Internet of Things with Security-by-Contract and Fog Computing. In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT). IEEE, Limerick, Ireland, pp. 1–6, Apr. 2019, ISBN: 978-1-5386-4980-0, URL: https://ieeexplore.ieee.org/document/8767243/, visited on: 10/08/2019.

[Go] Goodin, D.: Record-breaking DDoS reportedly delivered by >145k hacked cameras, https://arstechnica.com/information-technology/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/, Accessed: 2020-04-09.

[Ha] Hamza, A.: MUD Profiles, https://iotanalytics.unsw.edu.au/mudprofiles, Accessed: 2020-04-09.

[Ha18] Hamza, A. et al.: Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles. In: Proceedings of the 2018 Workshop on IoT Security and Privacy. IoT S&P '18, Association for Computing Machinery, Budapest, Hungary, pp. 8–14, 2018, ISBN: 9781450359054, URL: https://doi.org/10.1145/3229565.3229566.

[Hi] Hill, K.: Baby Monitor Hacker Still Terrorizing Babies And Their Parents, https://www.forbes.com/sites/kashmirhill/2014/04/29/baby-monitor-hacker-still-terrorizing-babies-and-their-parents/, Accessed: 2020-04-09.

[LDR19] Lear, E.; Droms, R.; Romascanu, D.: Manufacturer Usage Description Specification. Published: RFC 8520, RFC Editor, 2019.

[Ma] Matthíasson, G.: DBAC Device Detection, https://github.com/gdnoz/DBAC-Device-Detection, Accessed: 2020-04-09.

[Mi17] Miettinen, M. et al.: IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). Pp. 2177–2184, June 2017.

[Sa19] Samuel Marchal et al.: AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication - IEEE Journals & Magazine. IEEE Journal on Selected Areas in Communications 37/6, pp. 1402–1412, June 2019, URL: https://ieeexplore.ieee.org/abstract/document/8664655, visited on: 10/27/2019.

[Th19] Thomsen, M. D.: Device-Based Access Control, Accessed: 2020-04-09, MA thesis, Denmark: Danmarks Tekniske Universitet, 2019.