MUDThread - Securing Constrained IoT Networks via Manufacturer Usage Descriptions

Luke Houben*†, Thijs Terhoeve†, Savio Sciancalepore*
*Eindhoven University of Technology – Eindhoven, Netherlands
†Verano®, Eindhoven, Netherlands

Abstract—[1]**The Manufacturer Usage Description (MUD) standard recently published by the IETF allows manufacturers of Internet of Things (IoT) devices to equip their products with device specifications, i.e., information about the expected network connections of the devices. Such data can be used to detect unauthorized behavior and mitigate attacks involving IoT devices. However, at the time of this writing, no previous work integrated security services based on MUD into constrained IoT networks, e.g., the ones using the standardized protocol stack Thread. This paper proposes MUDThread, a framework for integrating and managing security services into constrained Thread-based IoT networks using MUD-derived security specifications. Using MUDThread, IoT devices can provide MUD-related information at the join of the network using a standard-compliant extension of the Mesh Link Establishment protocol. At the same time, the MUD Manager, integrated into the edge border router of the network, can enforce MUD-based rules to stop unauthorized network traffic. We deploy a Proof-of-Concept of our solution using actual nRF5340 and nRF 52833 IoT devices, and we experimentally verify its limited communication latency (0.012% more) and capability to detect both incoming and outgoing unauthorized network traffic during regular operations of a constrained IoT network.**

Index Terms—**IoT Security; Constrained Devices; IoT Firewall.**

I. Introduction

In the last years, we experienced the pervasive diffusion of Internet of Things (IoT) devices in homes, offices, and industrial environments, with up to 29 billion devices being deployed by 2030 [1]. At the same time, even more attacks in the wild exploit the limited processing, storage, and energy capabilities of such IoT devices to easily take over them and launch attacks. Notable examples of such attacks include Stuxnet (2010), and Mirai (2016) [2].

All such attacks exploit the IoT devices' misconfiguration to change their regular network behaviour, e.g., letting them act according to the commands received by a remote command-and-control server. Taking into account such considerations, various companies and stakeholders within the Internet Engineering Task Force (IETF) pushed the manufacturers of IoT devices to define precisely the expected network traffic profile of the devices, to ease the detection of possibly unauthorized connections. Such efforts resulted in the definition of the Manufacturer Usage Description (MUD) standard, allowing the manufacturers to specify, for each device, the expected outgoing and incoming network connections [3].

[1]This is a personal copy of the authors. Not for redistribution. The final version of the paper will be available soon through the digital library IEEExplore.

A few contributions in the last years proposed to integrate the MUD specification in the security management of various networking environments. For example, Hamza et al. [4] provide tools for allowing anyone to generate MUD files adhering to the IETF specification based on network traffic captures, Yadav et al. [5] applied MUD in a domestic IoT environment, Feraudo et al. worked on collaborative distributed learning in untrusted IoT environments [6] and on optimizing the parsing of MUD rules to alleviate network latency [7], while Morgese et al. [8] use MUD-rejected traffic from IoT devices in smart homes to identify network threats at large scale.

However, to the best of the authors' knowledge, no contributions specifically investigated the integration of MUD in very constrained networking environments, such as the ones using the state-of-the-art Thread protocol stack for constrained devices [9]. Several non-trivial additional challenges emerge in such scenarios, e.g., how to orchestrate security services in such constrained networks, enforce security services, and allow constrained devices to efficiently and timely deliver device specifications to the entity enforcing security measures.

Contribution. In this paper, we propose *MUDThread*, a framework integrating MUD-based security monitoring into constrained IoT networks running the standardized protocol stack Thread. *MUDThread* manages network traffic security by monitoring all incoming and outgoing connections to the IoT network via two dedicated processes, namely the *MUD Manager* and the *MUD Firewall*, integrated into the border router at the edge of the Thread-based IoT network. As a novel distinctive feature, constrained IoT devices provide information for retrieving the corresponding device specifications by integrating MUD-related information into extended Mesh Link Establishment (MLE) messages, delivered at the join of the network. We deployed a Proof-of-Concept (PoC) of our solution using actual constrained devices nRF5340 and nRF52833 and a border router Raspberry Pi 4B and released the code as open-source [10]. We also tested the performance of *MUDThread* in terms of overhead and attack detection rates in various configurations. We show that *MUDThread* can detect all unauthorized incoming and outgoing connections—deviating from the MUD profile of the devices—while posing minimal additional latency on network communications (only 0.012% of the overall network latency). To the best of our knowledge, *MUDThread* is the first framework defining and automating the deployment of MUD on very constrained IoT networks.

Roadmap. The paper is organized as follows. Sect. II provides preliminaries, Sect. III introduces the scenario and adversary model, Sect. IV discusses *MUDThread*, Sect. V provides results obtained through our experimental PoC and, finally, Sect. VI concludes the paper and outlines future work.

II. Background

Thread and OpenThread. Thread is a recently created publicly-available specification first released in 2015, aiming at creating an IoT protocol specification interoperable with traditional Internet Protocol (IP) networks [11]. Thread is a low-power, low-latency wireless mesh networking protocol stack composed of open and well-known standards, including: (i) Constrained Application Protocol (CoAP) for the application layer, (ii) User Datagram Protocol (UDP) for the transport layer, (iii) Routing Protocol for Low-Power and Lossy Networks (RPL) for the routing layer, (iV) IPv6 over Low-Power and Lossy Networks (6LoWPAN) for the adaptation layer, (v) MLE for network joining and maintenance, and (vi) IEEE 802.15.4 for the PHY and MAC layers. The Thread specification also describes the network components, including: (a) the *Thread Border Router*, which both enables the communication between Thread-based devices and connects the Thread network to the existing Local Area Network (LAN), (b) the *Thread routers*, responsible for handling incoming and outgoing packets to and from end devices and providing caching services, and (c) multiple resource-limited and energy-constrained *end IoT devices*, which turn on their radio only at specific times for communication, consuming the least amount of energy.

To accelerate deployment, Google recently released its own Thread implementation as open-source, namely *OpenThread*, that can be ported to many embedded operating systems [9]. Note that OpenThread only contains the Thread protocol stack. Since a border router also connects to the LAN, the OpenThread Border Router (OTBR) requires additional protocol stacks, e.g., IEEE 802.11 and IEEE 802.3. To this aim, Google also provides a certified open-source implementation for the OTBR.

Manufacturer Usage Description. MUD, defined in RFC 8520 [3], is a standard specification aimed at formally describing the network behavior of IoT appliances [3]. This is usually possible for IoT devices due to their limited functionalities. MUD allows the manufacturers of IoT devices to indicate the devices' expected outgoing and incoming connections through a MUD file. It is a JavaScript Object Notation (JSON) file containing the root object *ietf-mud*, reporting information about the intended usage of the device and metadata such as the version, file location, and file signature [3]. The file might also contain a field *ietf-access-control-list*, defining an Access Control List (ACL) with multiple Access Control Elements (ACEs), each indicating a service used by the device based on source/destination IP or port. The RFC 8520 also provides a few examples. By default, MUD rules can include addresses of DNS servers used to resolve IP addresses. At the same time, they do not provide information on the frequency of communication patterns and (expected) behaviors following specific interactions on the network.
The network administrators deploying the MUD-enabled device can download the MUD file at a given link, namely the MUD Uniform Resource Locator (URL). When a new device joins the network, the device should transmit the MUD URL to the entity enforcing access control rules. The standard specification defines three methods for this: (i) the Dynamic Host Configuration Protocol (DHCP); (ii) the Link Layer Discovery Protocol (LLDP); and (iii) via an extension of the X.509 certificate of the device. The specification also allows for providing digital signatures to authenticate the MUD URL. However, no standard mechanism is currently defined to bind a device identifier with the related MUD file. When IoT devices are not equipped with MUD files by default, network administrators can use tools like MUDgee to generate MUD-compliant files [4].
Although MUD was released in 2019, it has not yet been widely adopted. It does, however, have the potential to become a powerful tool to aid intrusion detection systems in preventing IoT attacks by ensuring that devices can only perform the functions intended by their vendor. Also, the scientific community [4] and some companies (e.g., Cisco, see https://github.com/CiscoDevNet/MUD-Manager) are providing tools for automatically generating MUD files based on the observed network traffic, favouring MUD adoption.

III. Scenario and Adversary Model

Scenario. We consider a generic IoT network, including several constrained IoT devices delivered by various manufacturers. Such devices feature limited computational, storage, communication, and energy resources. The devices exchange packets wirelessly on the communication frequency 2.4 GHz using the IEEE 802.15.4 communication technology. To interact outside of the local IoT network, the devices connect in a mesh network topology to a Border Router (BR), connected in turn to a *LAN Gateway Router* and to the Internet. The IoT devices run the protocol stack Thread (Sec. II), so integrating MLE to join the IoT network. The aim of our proposed framework *MUDThread* is to orchestrate the deployment of security services in this network, using the specifications provided by the IoT devices within MUD files.

Adversary and Threat Model. In this manuscript, we consider an adversary trying to use remote legitimate IoT devices exposed over the Internet to launch Denial of Service (DoS) attacks, similar to the Mirai malware [2]. Accordingly, we consider that the adversary is using malware that crawls the web for IoT devices, allowing remote connection on port 23 via the Telnet protocol. When the malware finds such devices, it performs a brute-force attack on Telnet credentials, using a pre-programmed list of recurring usernames and passwords to authenticate successfully. Upon successful authentication, the malware delivers the IP, username, and password to an external service in charge of loading malicious software on the device. Note that the adversary is fully remote, i.e., it does not have physical access to the devices. Also, it does not have access to the private keys of legitimate manufacturers, so not being able to forge MUD files. In Sec. V, we test the capability of *MUDThread* to identify and reject the described attack.

IV. MUDTHREAD

A. Network Architecture

MUDThread builds on the consideration that enforcing security via MUD on constrained IoT devices is unfeasible for many reasons. First, IoT devices can be very constrained regarding processing, memory, and energy resources, making it hard to enforce security processes locally. Also, in many network deployments, users cannot modify the code running on such devices, e.g., due to discontinued firmware and software updates. Even when this is possible, users could neglect to update the device. Moreover, users could decide not to install firmware upgrades on purpose to avoid service disruption and minimize short-term business losses. Finally, the adversary might gain complete control of the device, stopping the execution of any security-related process. Thus, *MUDThread* delegates the management of MUD-based traffic monitoring to the network BR. Fig. 1 shows the proposed network architecture.

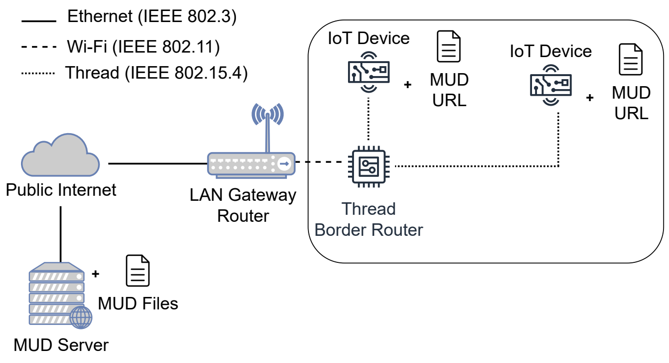

Fig. 1. Overview of the proposed *MUDThread* network architecture.

It includes the following entities: the *IoT devices*, the *Thread Border Router*, the *LAN Gateway Router* and the *MUD Server*. As discussed above, we consider constrained IoT devices running a firmware compliant with Thread (Sec. II). Thus, the manufacturer or the network administrator equips such IoT devices with a *MUD URL*, stored in the non-volatile memory, indicating where it is possible to download the *MUD File* of the device. We provide more details on the usage of such URLs and how IoT devices deliver related information in Sec. IV-B. The MUD URL redirects to the *MUD Server*. It is a file server hosted outside the LAN, managed by one or more manufacturers of IoT devices. Its purpose is publicly hosting MUD files to facilitate proper MUD file discovery. As per the MUD specification, the MUD file contains JSON entries defining a specific allowed network traffic to or from the IoT device. The MUD file also integrates a signature, used for authenticity and integrity. The *Thread Border Router* sits at the boundaries between the Thread IoT network and the LAN. The role of the Thread BR is to convert IEEE 802.15.4 packets into a suitable format and vice versa, depending on the source and destination of the packet. In line with many network deployments, we consider a wireless connection between the Thread BR and the LAN Gateway Router, e.g., via IEEE 802.11. On the Thread BR, we deploy two additional security entities: the *MUD Manager* and the *MUD Firewall*. The *MUD Manager* is the component responsible for managing security in the Thread network. It listens for incoming MUD URLs coming from IoT devices and, once received, it contacts the *MUD Server* to obtain the *MUD File* of the specific IoT device. It processes each MUD file entry to convert them to executable ACEs. The Thread BR also includes the *MUD Firewall*, i.e., a process responsible for enforcing the ACEs generated by the MUD Manager through an *allow-listing* approach. For each incoming (outgoing) packet to (from) the local IoT network, the MUD firewall checks if the source (destination) IP and port of such packet match any entry in the overall ACL. If so, it allows the packet to reach its intended destination. Otherwise, it drops the packet. It may also log such events locally. Finally, the connection between the Thread BR and the public Internet is provided through the *LAN Gateway Router*. It converts wireless IEEE 802.11 packets into IEEE 802.3 (Ethernet) packets, and viceversa. The LAN Gateway router and the Thread BR often coexist in the same device, e.g., a multi-protocol gateway. In such cases, the connection between the Thread BR and the LAN Gateway router is only logical (wired).

B. Protocol Flow

Fig. 2 shows the sequence of operations required to enable the MUD Firewall.

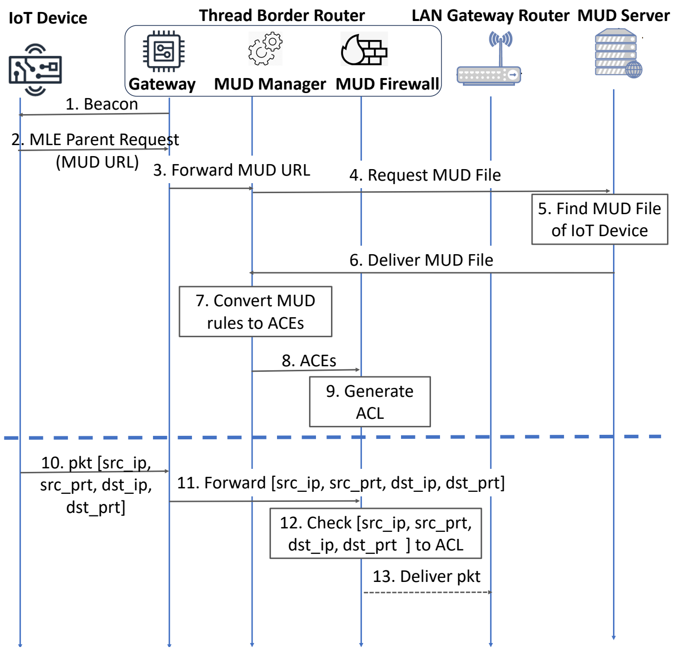
IoT Device
Thread Border Router
LAN Gateway Router
MUD Server
Gateway
MUD Manager
MUD Firewall
1. Beacon
2. MLE Parent Request (MUD URL)
3. Forward MUD URL
4. Request MUD File
5. Find MUD File of IoT Device
6. Deliver MUD File
7. Convert MUD rules to ACEs
8. ACEs
9. Generate ACL
10. pkt [src_ip, src_prt, dst_ip, dst_prt]
11. Forward [src_ip, src_prt, dst_ip, dst_prt]
12. Check [src_ip, src_prt, dst_ip, dst_prt] to ACL
13. Deliver pkt

Fig. 2. Sequence diagram of the operations required by *MUDThread* to create the ACL (top) and enforce security (bottom).

1) The Thread BR periodically broadcasts beacon messages, reporting identification and synchronization details of the Thread network.

2) At the reception of the beacon, the IoT device multicasts the MLE Parent Request message to *All Routers* in the Thread network to look for a suitable parent node for uplink traffic. The payload of the *MLE Parent Request* message includes the MUD URL of the device, optionally encrypted and authenticated via IEEE 802.15.4.
3) The Thread BR decodes the *MLE Parent Request*, extracts the MUD URL, and forwards it to the MUD Manager.
4) The MUD Manager checks the validity of the MUD URL. If it is valid, it uses it to contact the MUD Server, requesting the corresponding MUD File.
5) The MUD Server looks for the requested MUD File. If the file does not exist, it stops execution.
6) The MUD Server returns the MUD File to the MUD Manager.
7) The MUD Manager evaluates the freshness and validity of the MUD file and, if necessary, converts each entry in the file into an ACE.
8) The MUD Manager forwards the ACEs to the MUD Firewall.
9) The MUD Firewall inserts each ACE into the ACL.
10) At runtime, upon sensing, the IoT device delivers a data packet to the Thread BR, indicating a specific destination IP address and port.
11) The Thread BR extracts the source IP, source port, destination IP and destination port and forwards them to the MUD Firewall.
12) The MUD Firewall checks if any of the ACEs allows the IoT device with the given source IP and port to generate traffic towards the specific destination IP address using the specified destination port.
13) If an ACE exist, allowing for such traffic, the data packet is forwarded using the specific destination IP and port. Otherwise, the MUD Firewall rejects the packet. Note that operations similar to the latest two steps occur when the Thread BR receives a packet from the Internet to one of the IoT devices. In such a case, the data packet is forwarded to the IoT device only if an ACE on the MUD Firewall allows communication between the source IP, source port, destination IP and destination port specified in the packet.

Note that the IoT devices deliver the MUD URL in the MLE parent request only until joining to the Thread BR. The MUD Manager is responsible for keeping MUD files updated without further involvement by IoT devices. With the increased number of deployed IoT devices, additional ACEs may be added to the MUD firewall, increasing its complexity. The problem goes beyond *MUDThread*, and many different firewall engineering solutions are available in the literature [12]. All such solutions are compatible with *MUDThread* and can be integrated to improve efficiency.
Also, note that MUD files of a given device can change, e.g., due to a firmware update. At the reboot following such firmware updates, the MUD manager in *MUDThread* checks if an updated MUD file is available and possibly deploys it.

V. Implementation and Performance Evaluation

A. *Proof-of-Concept Implementation*

Fig. 3 depicts the architecture of our experimental PoC, using an actual IoT network.

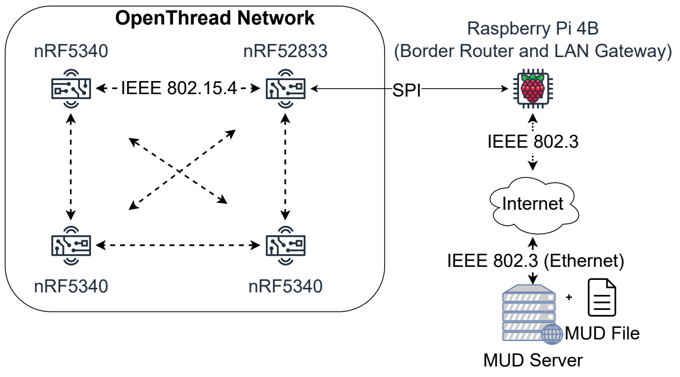

Fig. 3. PoC of *MUDThread*.

Hardware Details. We pick the board nRF5340 provided by Nordic Semiconductor as the IoT device of our PoC. It features a single ARM Cortex M-33 processor running at 64 MHz, 256 kB of Flash, 64 kB of SRAM, ultra-low-power modes for battery longevity and various wireless communication technologies, including Thread. As the Thread BR, we use a Raspberry Pi (RPi) model 4B. It is a tiny embedded computer equipped with a Cortex-A72 processor running at 1.5 GHz, featuring 4 GB of preinstalled Random Access Memory (RAM) and a micro-SD slot for storage. It also includes an IEEE 802.11b/g/n interface, allowing it to serve as the LAN Gateway Router. As the RPi does not feature a Thread-compatible module, we connect to the RPi the board nRF52833 provided by Nordic Semiconductor. This board features a single ARM Cortex M-33 processor running at 64 MHz, 512 kB of Flash, 128 kB of SRAM, and Thread communication capabilities. Finally, as for the MUD server, we deployed a simple web server using *NGINX* on a remotely located Virtual Private Server (VPS) running Linux Ubuntu 20.04.

Software Details. We used OpenThread as the Thread-based Operative System (OS) running on both the nRF5340 and the nRF52840 IoT device. Within OpenThread, we keep the default configuration and modify the default implementation of the MLE protocol in a standard-compliant way, to allow the IoT device to include the MUD URL in the MLE Parent Request. In particular, we created a new Type-Length Value (TLV) with Type 27 to convey the MUD URL. On the Thread gateway, we extended the default OTBR implementation with the code necessary to extract the MUD URL and deliver it to the MUD Manager. The communication between the Thread Gateway and the MUD Manager (hosted on the Raspberry Pi) uses the protocol SPI [13]. Moreover, the Raspberry Pi runs the Raspian OS. Finally, the MUD server can be reached only via HTTPs by browsing for the URL https://mud.verano.nl/. We generated MUD files for all IoT devices and made them available at a specific MUD URL. Interested readers can find an example MUD File at https://mud.verano.nl/example/

mud.json. To emulate an actual IoT network deployment, we let the IoT devices exchange traffic according to a real-world use case, i.e., a deployment of smart solar panels provided by Verano. To this aim, we consider that all domain addresses have already been resolved, so as to work with pre-resolved IP addresses. Finally, as per the MUD Firewall, we use the tool *ip6tables*, enabled by default on the Raspberry Pi. This firewall can filter IPv6 packets coming in on the host device. Using the allow rules from the MUD Manager, the tool creates an executable bash script containing *ip6tables* commands to set the firewall chains accordingly. We release the source code of *MUDThread* as open-source at [10].

B. Experiments

We run several experiments to evaluate the overhead and performance of *MUDThread*.

We first evaluate the overhead of *MUDThread* in terms of additional network latency. To this aim, we let one IoT device deliver $10,000$ ICMP Echo Requests packets (allowed according to the related MUD file). We record the Round Trip Time (RTT) of the packets with and without *MUDThread* and *netfilter6* active to evaluate the latency introduced by our solution. Fig. 4 summarizes our results. Without *MUDThread*

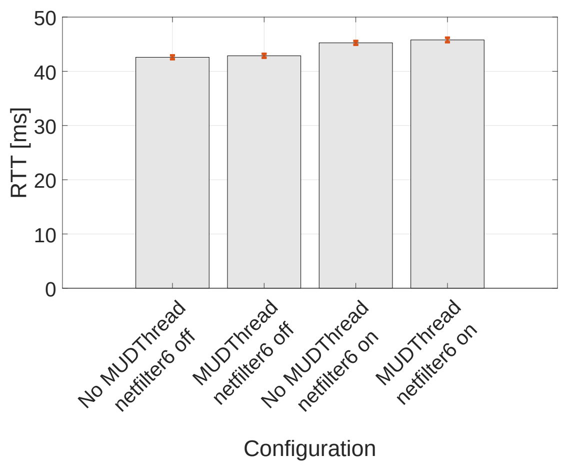

Fig. 4. RTT Overhead of *MUDThread*.

and *netfilter6*, the average RTT value is 42.59 ms. Such value grows to 45.24 ms by only activating *netfilter6* and finally amounts to 45.79 ms by also enabling *MUDThread*, so reporting an overhead of only $\approx 1.2\%$. Such results confirm the viability of our approach, as it affects the network latency negligibly. Consider that, according to IEEE 802.15.4, Thread-based IoT networks with a single border router can transmit up to one packet every time slot, with a time slot lasting no less than 10 ms [14], i.e., approximately the same as the average latency experimentally measured. Also, consider that the devices do not use all the time slots to transmit/receive packets due to energy consumption constraints and that similar ACEs for multiple IoT nodes can be grouped, leading to a sub-linear latency increase with the number of nodes (see [12] for an overview of firewall engineering solutions). Thus, even in the worst case where all devices connected to the network have different MUD files, we expect the network size not to generate significant scalability issues. To provide further insights into such aspects, we measured the RTT of $1,000$ packets when deploying multiple MUD entries on the MUD Firewall, i.e., 10, 100, and 1,000 rules, respectively. We configured such rules in the worst-case scenario, i.e., the entry applying to our IoT device is always the last in the chain. We report in Fig. 5 the median and 99 percentiles of all our tests. We notice that the increase in the number of entries

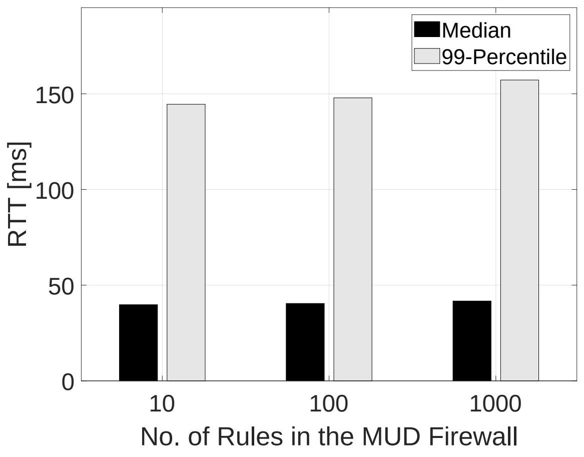

Fig. 5. RTT Overhead of *MUDThread* when the MUD firewall is configured with various rules.

in the MUD firewall causes a slight increase in the median RTT. With 10 rules, the median RTT is 39.79 ms, increasing to 40.45 ms with 100 rules and 41.72 ms with $1,000$ rules. Accordingly, also the 99-percentiles increase from 144.52 ms with 10 rules to 157.18 ms with $1,000$ rules. Still, such values remain limited, confirming the viability of our approach.

We also test the correct functionality of *MUDThread*, i.e., the capability to stop attacks exploiting anomalous network traffic while allowing legitimate traffic. To this aim, we took inspiration from the Mirai malware [2]. Mirai could spread quickly because of weak credentials on IoT devices and networks, which is an issue relevant as of today [15]. In particular, the hosts infected by Mirai were chosen by choosing random IP addresses. Once a publicly available IoT device was identified, the Mirai malware brute-forced the Telnet credentials. Telnet runs by default on port 23. Mirai used a pre-programmed list of recurring usernames and passwords to authenticate to the host over Telnet. If the host responded and authentication was successful, the IP, username, and password were sent to another service in charge of uploading malicious software. Such software allows to keep control over the device, even with dynamic IP addressing in place. Although many IoT devices block incoming Telnet connections by default today, Mirai still uses the same approach today to target IoT devices in the wild. We use the same attack model of Mirai to test the capability of our approach to reject connections falling outside their MUD profile, independent of whether such attempts are successful or not. Thus, we generated a script that, on input a given IP address, runs a brute-force attack on the Telnet credentials. Then, we test the capability of *MUDThread* to detect and reject such a brute-force attack. We also mix such (malicious) traffic with benign traffic, i.e., ICMP Echo Requests. Our solution successfully blocks all Telnet connections to the IoT device, since Telnet traffic is not allowed to that device. Also, 100% of the ICMP echo requests are correctly delivered over the Internet.

Finally, we test *MUDThread* in a scenario where the end IoT device is compromised and tries to deliver custom network traffic to an IP address outside the local IoT network. This test aims to verify the capability of *MUDThread* to detect and reject malicious traffic flowing out of the IoT network. Also, it considers a scenario where *MUDThread* is deployed after the IoT network has been operational for some time, and thus, some devices may have already been compromised. To this aim, we instruct the end IoT device to deliver several ICMP echo requests to an IP address randomly chosen outside the local IoT network with an increasing packet injection rate, i.e., 1, 10, and 100 packets/sec. Note that, in this experiment, we look at the effect of the traffic rate on *MUDThread*, independently of how many devices generate the traffic. We test our PoC without and with *MUDThread*, respectively, and report the average results in Tab. I. For all the tested configurations, when

TABLE I
RESULTS OF EXPERIMENT 3: UNAUTHORIZED OUTGOING PACKETS.

Injection Rate [pkt/sec]	1		10		100	
***MUDThread* Active**	No	Yes	No	Yes	No	Yes
Packets To send	500	500	5,000	5,000	50,000	50,000
Packets received at the MUD Firewall	500	500	5,000	5,000	$48,640$	$49,155$
Discarded Packets	0	500	0	5,000	0	49,155
% Discarded Packets	0	100	0	100	0	100

active, *MUDThread* correctly identifies anomalous traffic. The detection occurs as the destination IP address is not in the MUD file corresponding to the device, allowing *MUDThread* to identify and reject such packets. We also notice that, by increasing the packet injection rate, some packets are dropped (with 100 packets/sec, we instructed the IoT device to send $50,000$ packets, but on average, $48,640$ and $49,155$ were effectively delivered without and with *MUDThread* in place, respectively). We observe such phenomena both with and without *MUDThread*, and they are attributable solely to the congestion of the IEEE 802.15.4 communication link. These experimental findings further confirm that *MUDThread* does not generate scalability issues.

Therefore, *MUDThread* can be integrated seamlessly into an IoT network with minimal overhead in terms of network latency. Also, *MUDThread* can effectively block all unauthorized traffic of the constrained IoT network.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed *MUDThread*, a framework for enforcing security services at the edge of constrained Thread-based Internet of Things networks. As a distinctive feature, *MUDThread* leverages MUD profiles, i.e., specifications provided by the manufacturers of IoT devices describing the expected traffic flows of such devices. Taking such information into account, *MUDThread* defines the protocol flow necessary to convert MUD files into actionable ACEs, enforced by the MUD manager on the Border Router at the edge of the local IoT network. We deployed a Proof-of-Concept of *MUDThread* using actual nRF5340 and nRF52833 IoT devices using the OpenThread (OT) protocol stack and a Raspberry Pi 4B as the OTBR, and we tested extensively the performance of our solution, both in terms of incurred overhead and security guarantees. We demonstrate that leveraging MUD files, *MUDThread* can detect any traffic flow falling outside of the expected behavior of the IoT device (both incoming and outgoing traffic flows) while posing a minimal toll on network communication performance. Our future work will focus on integrating *MUDThread* into complex network topologies including multi-OTBR networks in extensive operational facilities.

ACKNOWLEDGEMENTS

This work has been supported by the INTERSECT project, Grant No. NWA.1162.18.301, funded by the Netherlands Organisation for Scientific Research (NWO). Any opinions, findings, conclusions, or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of NWO.

REFERENCES

[1] D. Wang, D. Chen, B. Song, et al., "From IoT to 5G I-IoT: The next generation IoT-based intelligent algorithms and 5G technologies," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 114–120, 2018.
[2] M. Antonakakis, T. April, M. Bailey, et al., "Understanding the Mirai Botnet," in *26th {USENIX} Security Symposium*, 2017, pp. 1093–1110.
[3] E. Lear et al., "Rfc 8520: Manufacturer usage description specification," https://www.rfc-editor.org/rfc/rfc8520, accessed: 9-Apr-2024.
[4] A. Hamza, D. Ranathunga, H. Gharakheili, et al., "Clear as MUD: Generating, validating and applying IoT behavioral profiles," in *Proc. Workshop on IoT Security and Privacy*, 2018, pp. 8–14.
[5] P. Yadav, V. Safronov, and R. Mortier, "Enforcing Accountability in Smart Built-in IoT Environment using MUD," in *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, 2019, pp. 368–369.
[6] A. Feraudo, P. Yadav, V. Safronov, et al., "CoLearn: Enabling federated learning in MUD-compliant IoT edge networks," in *Proc. ACM International Workshop on Edge Systems, Analytics and Networking*, 2020, pp. 25–30.
[7] A. Feraudo, D. Popescu, A. Diana, et al., "Mitigating IoT Botnet DDos Attacks through MUD and eBPF based Traffic Filtering," in *ACM International Conference on Distributed Computing and Networking*, 2023.
[8] L. Morgese Zangrandi, T. Van Ede, T. Booij, S. Sciancalepore, L. Allodi, and A. Continella, "Stepping out of the MUD: Contextual threat information for IoT devices with manufacturer-provided behavior profiles," in *Proc. Annual Computer Security Applications Conference*, 2022, pp. 467–480.
[9] H.-S. Kim, S. Kumar, and D. E. Culler, "Thread/OpenThread: A Compromise in Low-Power Wireless Multihop Network Architecture for the Internet of Things," *IEEE Communications Magazine*, vol. 57, no. 7, pp. 55–61, 2019.
[10] L. Houben, T. Terhoeve, and S. Sciancalepore, "Open Source Code of MUDThread," https://github.com/LukeHouben/ot-mudthread, accessed: 9-Apr-2024.
[11] Thread Group, "Thread 1.3.0 Specification," https://www.threadgroup.org/ThreadSpec, accessed: 9-Apr-2024.
[12] A. Voronkov, L. Iwaya, L. Martucci, et al., "Systematic literature review on usability of firewall configuration," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–35, 2017.

[13] R. Quattlebaum and J. Woodyatt, “Spinel Host-Controller Protocol,” Internet Engineering Task Force, Internet-Draft draft-rquattle-spinel-unified-00, May 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-rquattle-spinel-unified/00/

[14] S. Sciancalepore, P. Tedeschi, U. Riasat, and R. Di Pietro, “Mitigating Energy Depletion Attacks in IoT via Random Time-Slotted Channel Access,” in *IEEE Conf. on Communications and Network Security*. IEEE, 2021, pp. 10–18.

[15] S. Furnell, “Assessing website password practices–Unchanged after fifteen years?” *Computers & Security*, vol. 120, p. 102790, 2022.

Biographies

Luke Houben is a Master’s graduate at TU/e, Eindhoven, specializing in information security. After graduating in 2023 from TU/e in Eindhoven, Netherlands, he started working as a project manager at CodeFlex. His interests lie in the fields of IoT and network security, cryptography, and their implementations.

Thijs Terhoeve , with a background in Mechanical Engineering, has seamlessly integrated this knowledge with passions for Electrical Engineering and Computer Science. This unique blend has propelled him through various IT roles across multiple firms, where he consistently innovated at the intersection of mechanics, electronics, and software. Lauded for his analytical mindset and adaptability, Thijs continues to pioneer advancements that bridge traditional and digital engineering realms.

Savio Sciancalepore is Assistant Professor at TU/e, Eindhoven, Netherlands. He received the PhD degree in 2017 from Politecnico di Bari, Italy. From 2017 to 2020, he was Postdoctoral researcher at HBKU, Doha, Qatar. His research interests are in network security and privacy in IoT, Mobile and Wireless Networks.