A Gateway-based MUD Architecture to Enhance Smart Home Security

A Gateway-based MUD Architecture to Enhance Smart Home Security

Fulvio Corno
Dipartimento di Automatica e Informatica
Politecnico di Torino
Turin, Italy
fulvio.corno@polito.it

Luca Mannella
Dipartimento di Automatica e Informatica
Politecnico di Torino
Turin, Italy
luca.mannella@polito.it

***Abstract*—Smart home systems, including consumer-grade Internet of Things (IoT) devices, are in a dangerous situation. On the one hand, the number of smart homes is increasing. On the other hand, the devices in these dwellings are often affected by vulnerabilities that could be exploited to generate massive (distributed) attacks. To mitigate the issue of having compromised devices involved in such attacks, the Internet Engineering Task Force (IETF) recently proposed a new standard: the Manufacturer Usage Description (MUD).**

The main contribution of this paper is to propose a slightly extended version of the MUD architecture. This architecture is centered around a smart home gateway (SHG) that can be extended through the contributions of plug-in developers. Indeed, our proposed approach allows developers to specify which endpoints their plug-ins need to reach. These requirements will then be processed to generate a consolidated gateway-level MUD file exposed by the SHG itself. Thus, thanks to this solution and developers' intervention, even devices that are not natively "MUD-enabled" would be protected by the MUD standard if integrated through a proper plug-in. Moreover, these requirements are transparent for the device itself.

To demonstrate the feasibility of this approach, we realized a proof-of-concept for a widespread open-source smart home gateway: Home Assistant.

***Index Terms*—Cybersecurity, Gateways, Home Assistant, Internet of Things, Manufacturer Usage Description, Smart Home**

I. Introduction

According to the recent report published by IoT Analytics [1], despite the chip shortage in recent months, the Internet of Things (IoT) domain continues to grow. In 2021, they counted 12.2 billion active endpoints, and they forecast that in 2025 there will be approximately 27 billion connected IoT devices.

Within the IoT domains, in the last years, many researchers started to study smart homes. Nonetheless, they used different definitions of this concept in their work [2]. In one of the first surveys on smart home (SH) research [3], a SH was defined as "a dwelling incorporating a communications network that connects the key electrical appliances and services, and allows them to be remotely controlled, monitored or accessed".

This work was partially supported by Fondazione CRT (Cassa di Risparmio di Torino) and by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

However, even if this environment was studied, analyzed, and deployed for many years, SH systems are still in a critical situation. Indeed, while these systems are becoming more and more widespread, the IoT devices part of SHs are still affected by security vulnerabilities. To have an objective view of this situation, Kumar et al. [4] analyzed a considerable number of real-world houses (around 16 million dwellings from all around the world). Their research discovered that more than 40% of global houses have at least one IoT device (the number rises around 70% considering only North American homes). Analyzing the data collected from these devices, the authors disclosed that a significant percentage of the involved dwellings have an IoT device affected by at least one known vulnerability. Reasonably, the number of vulnerable devices increases if we consider that many vulnerabilities are yet to be discovered (or discovered and not revealed to producers). Furthermore, it is noteworthy that a compromised device could impact all the other machines connected to the same network [5] or could be involved in massive Distributed Denial of Service (DDoS) attacks like the famous Mirai Botnet [6].

To mitigate the issue of involving devices in such attacks, the Internet Engineering Task Force (IETF) recently proposed a new standard: the Manufacturer Usage Description (MUD) [7]. This standard is essentially based on a white-listing approach. It defines an architecture and a data model to restrict communication to/from IoT objects. In line with the standard's name, the device manufacturer has to specify the expected use of its device in a dedicated file (the MUD file). Unfortunately, even if this standard is considered promising by ENISA [8] and NIST [9], it is currently not yet widely deployed in real-world scenarios [10]. Indeed, many manufacturers do not produce MUD files yet. For this reason, many researchers proposed to introduce a third party able to generate such files in place of the manufacturers [10]. Among them, a smart home gateway (SHG) could be suitable to carry out this job.

SHGs are software solutions designed to manage and coordinate many smart homes' devices. Thanks to their privileged position, SHGs can even implement some additional security features for the home network [5], [11]. Indeed, they can add a further layer between the object and the network. The integration of each device with the SHG has to be developed by one of these three entities: the SHG *development team*,

the devices' *manufacturers*, or a *programmer* interested in using a particular object in its SH. The more devices and functionalities an SHG integrates, the more the community is interested in the project. Hence, to help developers and manufacturers to incorporate more smart objects possible, it is quite common for SHG teams to create gateways that are extensible through plug-ins (e.g., Home Assistant [12], OpenHAB [13], WebThings [14], etc.).

Considering what we discussed so far, the goal of this paper is to propose a slightly extended MUD architecture for allowing plug-in programmers to specify a set of MUD-compliant requirements. The SHG will then use these requirements to automatically generate a gateway-level MUD file in place of the original ones from the manufacturers.

To demonstrate the feasibility of this solution, we realized a proof-of-concept (PoC) for a widespread open-source SHG: Home Assistant (HAss). Currently, HAss does not handle the MUD standard at all. Therefore, we realized a HAss plug-in able to create the MUD file starting from a template (provided together with the proposed plug-in). To enforce the MUD policies, it is required the deployment of a MUD manager in the same network where the HAss instance is deployed.

The rest of the paper is structured as follows: Section II analyzes some related work on SH security and MUD. Section III provides more details about the MUD architecture and data model. Then, Section IV presents our proposed SHG-based architecture and the implemented PoC, while Section V discusses their possible limitations. Finally, Section VI concludes the paper and proposes insights into future activities.

II. Related Work

Several researchers have demonstrated how a compromised device could be used to affect others. For example, Kafle et al. [5] executed a lateral privilege escalation attack for disabling a Google's Nest Security Camera via a compromised Philips Hue. Moreover, a compromised object can even try to attack endpoints very far from its location. The massive Distributed Denial of Service (DDoS) attacks executed through botnets [15] are a perfect example of this possibility. Among them, we can not ignore the relevant impact of the famous Mirai Botnet, and all the other botnets related to it [6].

In addition, the code to deploy these objects in smart homes is often developed by hobbyist programmers. Their projects could present security issues that could affect the whole smart home [16]. This is also stressed by a recently published threat model for extensible smart home gateways (SHG), which highlights how dangerous undesired (or unpredicted) interaction among components can be [17]. However, since an SHG interacts with almost all the devices in a dwelling's network, it can be the perfect point for introducing an additional line of defense. For instance, Yang et Al. [18] proposed an SHG able to use onion routing to transmit IoT data streams, while Gajewski et al. [11] empowered a gateway with an Intrusion Detection System (IDS).

However, even if many ways to detect malicious traffic from IoT devices exist (e.g., using machine learning techniques [19]), a standardized approach to control these objects' behaviors can promote a secure and automated deployment (and management) of secure IoT solutions. In line with this concept, IETF proposed in 2019 the Manufacturer Usage Description (MUD) standard [7]. Thanks to this standard, manufacturers can define for their device the allowed endpoints for establishing or receiving communications. To achieve this goal, manufacturers have to write in a specific file (the MUD file) a list of policies (also called Access Control Lists). These policies declare a set of endpoints that the device must be able to reach to carry on its regular activities. The MUD files must be *securely* stored in a server under the manufacturer's control. By reading and processing these MUD files, a dedicated component on the home network (called MUD manager) can configure the network to enforce the specified MUD policies. Once the policies are active, only a certain number of connections is allowed for each IoT device compliant with such a standard. Reducing the allowed communications, MUD automatically reduced the possible attack surface.

In the last years, MUD has attracted many researchers [10]. It was considered a promising solution even by the National Institute of Standards and Technology (NIST) [9] — to mitigate security threats and to cope with Denial of Service (DoS) attacks — and by the European Union Agency for Cybersecurity (ENISA) — including MUD in their "Good practices for security of IoT" [8]. Moreover, some scholars proposed to expand or modify the MUD standard. For instance, Jin et al. [20] have filed a patent that extends MUD by considering some dynamic security aspects of smart buildings. Instead, Sajjad et al. [21] proposed enhancements to MUD architecture for identifying (and eliminating) the configuration vulnerabilities before creating the devices' MUD profiles.

A very comprehensive survey on the MUD was written by Hernández-Ramos et al. [10]. In their paper, the authors provide an accurate description of the MUD standard, a complete overview of the state-of-the-art, and some attractive research directions. They classified the research activities on the MUD into four different macro-areas: MUD profile *generation*, MUD profile *obtaining*, MUD profile *enforcement*, and *MUD-based application*. Our proposed approach can be placed inside the first category. Indeed, our method is an automatic generation (and exposure) of a MUD file that requires manual plug-in developers' intervention.

III. Manufacturer Usage Description (MUD)

The Request for Comments (RFC) of the Manufacturer Usage Description (MUD) [7] defines the following architectural components (graphically represented in Figure 1):

- The **MUD file**: a file describing a *Thing* and specifying which are the allowed endpoints for establishing communications (in both directions). It is written using JavaScript Object Notation (JSON) [22] and describes a set of policies. These policies are expressed using the YANG (Yet Another Next Generation) standard [23].
- The **MUD file server**: a server hosting the *MUD files*.

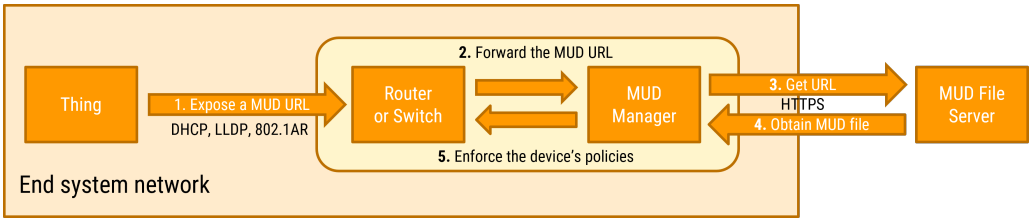

Fig. 1. Workflow of a standard MUD architecture.

- The **MUD manager**: a system that requests and receives the *MUD file* from the MUD server. After it has processed a MUD file, it directs operational or configuration changes to relevant network elements.
- The **MUD URL**: a URL used by the *MUD manager* to receive the *MUD file*.
- The **Thing**: an IoT device emitting a *MUD URL*.
- The **manufacturer**: an entity that configures the *Thing* to emit the *MUD URL*. It also asserts the recommendations in the *MUD file*. It might not always be the entity that constructs the device. E.g., it could be a systems integrator or a component provider.

Notably, the MUD file of a specific IoT device is not stored on the device itself. Instead, the device must inform the MUD manager where the MUD file can be retrieved. To provide this information, the Thing provides the MUD manager with a URL. This information can be spread using one of these protocols: DHCP [24], LLDP [25], or 802.1AR [26]. Via this URL, the MUD manager retrieves the MUD file through HTTPS [27] and communicate with the router (or with the proper network element) to enforce the enclosed policies.

How the policies are instantiated is not strictly described in the standard, but it depends on the implementation of the MUD manager and on the local network administrator. To provide some examples, the open source MUD Manager (osMUD) [28] enforces the policies through `iptables` [10] (a program able to configure the packet filter rules of the Linux kernel firewall [29]), while the NIST MUD manager [10] relies on Software Defined Networking (SDN) [30] (involving switches and controllers that observe the OpenFlow protocol [31]).

It is even possible that the MUD manager and the router (or whatever element has to enforce the policies) are embedded on the same physical device. Figure 1 visually describes the workflow of the previously described architecture.

The introduction of this standard aims to achieve a set of improvements in IoT security [7]. The first one is to reduce the device's threat surface. Indeed, only the endpoints specified by the manufacturer can communicate with the related device. The second one is to reduce the reaction time when particular vulnerabilities are discovered. Creating and deploying a patch for software running on many constrained IoT devices could be difficult [32]. Updating a single file stored in a single place under the complete control of the manufacturer would grant to act promptly for many devices. However, MUD should not be considered a replacement for traditional patches; it is only a tool to quickly create a countermeasure (while the patch is under development). Moreover, by reading the MUD file, it is easy to understand what are the endpoints with which a device needs to communicate (without the necessity of conducting a network traffic analysis to discover them).

IV. Proposed Approach

Considering that the MUD standard is not widely deployed yet, many scholars proposed different approaches to *generate*, *obtain*, and *enforce* the rules described in a MUD file [10]. Specifically, our paper proposes a new MUD semi-automatic generation methodology. Our proposal is based on the assumption that every smart home gateway (SHG) plug-in can benefit from the MUD (either in the case the plug-in integrates only new software functionalities) increasing the overall security of the SHG and, consequently, the whole smart home's security. Moreover, offering plug-in developers the chance to act in place of device manufacturers can increase the number of devices supported by the MUD standard inside a smart home.

To demonstrate the feasibility of the described architecture, we realized a proof-of-concept (PoC) based on an open-source implementation of the MUD architecture, suitably extended with our proposed functionality. The main components of this architecture (and their interactions) are represented in Figure 2. In detail, they are the following:

- **OpenWrt** [33]: an open-source Linux embedded system designed for routers;
- **osMUD** [28]: an open-source MUD manager designed to be installed on an OpenWrt-based router;
- **Home Assistant** [12]: an open-source Python-based extensible smart home gateway;
- **HAss MUD Integration**: a Home Assistant integration designed to create and expose a gateway-level MUD file;
- The local gateway-level generated **MUD file**: a MUD file obtained blending together all the requirements specified by each integration;
- The **MUD policies**: a set of requirements specified by the plug-in developer in a MUD-compliant way (the file containing these policies is also called MUD snippet).

Home Assistant can be extended through two different types of plug-in: integrations and add-ons. Integrations are more similar to traditional plug-ins (indeed, they are installed in HAss's core). On the contrary, add-ons are basically Docker containers. Therefore, in our PoC, to prove the feasibility of the approach, we considered only the first ones.

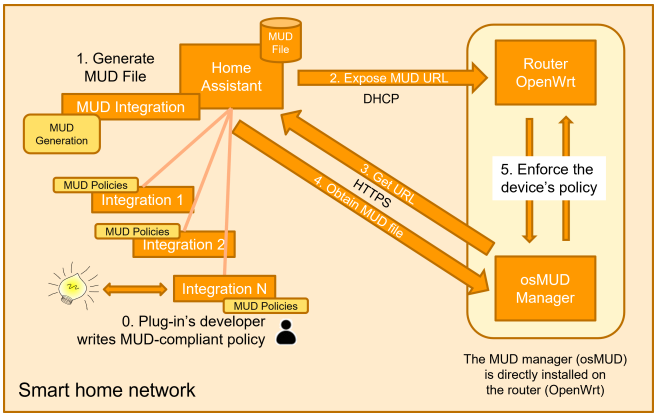

Fig. 2. A schema of the implemented proof-of-concept architecture.

When Home Assistant is started, it executes all the configured integrations and add-ons. At that moment, even the MUD integration starts and queries all the other integrations looking for the developers' requirements. Each integration have to specify its requirements in a dedicated JSON file. Once all the requirements are collected, the MUD integration unites them in a local MUD file (created starting from a predefined template). Then, the generated MUD file is signed using the SHG's private key. Once signed, the MUD integration stores that file in a dedicated folder (exposed by the Home Assistant web server) and informs the MUD manager (step 1).

In our PoC, we used an OpenWrt router [33] enabled to execute an open-source MUD manager called *osMUD* [28]. Among the notification approaches defined in MUD RFC [7], osMUD currently supports the Dynamic Host Configuration Protocol (DHCP) [24] only. Therefore, our HAss integration exposes the MUD file sending a *DHCP Request* to the router, including the proper MUD URL (step 2). This URL points to the HAss instance, which has, by default, a web server able to expose the generated file (the same web server also exposes the HAss dashboard). Once osMUD retrieves the MUD file (step 4), it verifies the associated signature and starts enforcing the enclosed policies. These policies are implemented by configuring the firewall of OpenWrt, `iptables` (step 5).

When users add new integrations to Home Assistant, to start them, they are forced to restart the SHG. Therefore, every time a new integration is added, the previously described process it is re-executed to keep the MUD file updated with all the available MUD-enabled integrations. When osMUD retrieves the new MUD file, it removes the old associated firewall rules and instantiates the new policies.

Unfortunately, when we started the implementation of our PoC, osMUD did not manage the DHCP requests that arrived from already known devices (they were simply ignored). In other words, the process of retrieving and enforcing the MUD policies of a device was started only if the MUD URL was stored in the first DHCP message sent to the MUD manager by that specific device. Hence, we extended osMUD's source code to properly process every request sent by each device. In our modified version, if osMUD receives a request from a device already known, it verifies if the associated MUD file was changed in the elapsed time. If so, it removes the old policies and enforces the new ones. We plan to open a pull request on the repository of osMUD to include our improvement in the MUD manager's default version.

To work properly, our architecture needs developers' contributions. Indeed, developers have to specify the endpoints their integrations want to reach (or to allow to be reached). This has to be done by writing a MUD-like file for each integration. Specifically, in this file, developers must write the objects `from-device-policy`, `to-device-policy` according to the MUD [7] and YANG standards [23]. Developers without a deep knowledge of the previously cited standards can use a MUD generation tool to have help in creating their snippets. For instance, they can use MUD Maker[1].

A. Experimental Results

To test our proof-of-concept (PoC), we created a dedicated local network inside our laboratory based on the schema reported in Figure 2. In this network, we deployed on two different Raspberry Pi 3B[2] the MUD manager (osMUD) and the smart home gateway (Home Assistant).

The Raspberry hosting the *MUD manager* has an OpenWrt operating system (OS), a Linux-based OS designed for routers. We used the version suggested by osMUD (17.01.6). As previously discussed, to interact correctly with our Home Assistant integration, we extended the original osMUD software. Specifically, we had to add the feature of processing MUD files even if they come from devices that already exposed them.

The Raspberry hosting the SHG was equipped with Home Assistant OS (version 9.2), a Linux-based operating system optimized to host HAss and its add-ons, and our HAss MUD integration. Then, we extended three installed integrations by providing them with a MUD snippet file:

- A **weather** integration: that requires reaching the website offering the weather's information.
- A custom emulated **temperature sensor** integration: that can store on a remote server the recorded value (and retrieve the last stored value).
- A custom emulated **switch** integration: that needs to reach a server to be remotely controlled.

These integrations require, in their MUD snippets, to access an endpoint in both directions. Once started, our HAss MUD integration collected the requirements and created a consolidated gateway-level MUD file. Whenever this MUD file is generated, the integration stores it in a dedicated folder (overriding the old version, if present) publicly exposed by the default HAss web server.

In our PoC, the generated MUD file contains eight policies (two policies for each involved integration, plus two stored in the template used as the starting point for forging the file).

[1]https://mudmaker.org, last visited on March 17th, 2023.

[2]https://www.raspberrypi.com/products/raspberry-pi-3-model-b/, last visited on March 17th, 2023.

These policies are then enforced into ten firewall rules (one for each MUD policy, plus two for blocking all the other TCP/IP flows in both directions).

We observed that the HAss MUD integration correctly generated, signed, and exposed the gateway-level MUD file. Then, the contained policies were successfully instantiated by osMUD. To verify the capability of the MUD integration to regenerate and re-expose the MUD file, we executed the process after adding one by one each of the previously cited integrations. Firstly, we verified that the integrations were still working correctly after the enforcement of the MUD policies. Then, we changed the target endpoints of the custom integrations, and we observed that the endpoints specified in the MUD files were properly reachable by these integrations, while other URLs were not.

To conclude, our PoC demonstrates that it is possible to automatically generate a MUD file at run-time and that our solution is perfectly compatible with a traditional MUD infrastructure.

V. Discussion and Limitations

The presented architecture can improve the security of every house equipped with an extensible smart home gateway (SHG). Indeed, allowing developers to specify requirements in place of the original manufacturer can increase the number of IoT devices covered by this standard. Besides, having an additional layer of protection for every SHG plug-in (even for plug-ins introducing new software functionalities only) can enhance the overall gateway security. Furthermore, having a global MUD file locally stored in the SHG can guarantee the enforcement of the MUD policies even if manufacturers' servers are unreachable (even in case devices or plug-ins are no longer supported).

The reported implementation and its experimental results proved the feasibility of the approach. However, in the current prototypical state, we may still identify the following limitations. Considering that our proof-of-concept (PoC) is based on the MUD architecture, it inherits all this standard's issues. For instance, malicious users could tamper with the deployed MUD policies on the OpenWrt router. Anyone having access to the router could compromise these policies; therefore, it must be adequately protected. Another possible issue is tampering with the MUD URL. If malicious users compromise that URL, they can decide what policies will be enforced on the network. If the reader wants to go deeper into the limits of the "plain MUD", we suggest reading [10].

Discussing the proposed PoC's specific limitation, the following are the most relevant. Firstly, it is essential to detect conflicts in user-defined policies. Suppose some integration developers decide to explicitly forbid access to a specific endpoint necessary for the execution of another integration. In that case, our *HAss MUD integration* has to decide which policy must be enforced. It is also important to establish how to manage integrations that do not have any explicit policy. Moreover, some developers could describe the same policy or declare overlapping policies. These issues were left out of this paper's scope and will be addressed in future work.

Another critical issue is the impact of the "MUD snippets" in our HAss integration. Anyone accessing the integrations source code could alter it (and alter the snippets). To mitigate this issue, following the approach of different stores, Home Assistant should force developers to sign their plug-ins (and executes only signed integrations and add-ons). A proposal restricted to our proposed architecture, could be to have a list of "authorized HAss-MUD developers". Through such a list, our integration could process only the MUD policies specified by "trusted" developers.

Moreover, once the MUD file is composed, to send the MUD URL, our HAss MUD integration uses DHCP over Internet Protocol (IP) v4. At the current time, this is the only protocol supported by osMUD for obtaining the MUD URL. We expect that osMUD will support more robust protocols in the following versions (i.e., the possibility of embedding the URL in an X.509 certificate [26]).

VI. Conclusions

This paper's main contribution is to present a slightly extended version of the conventional MUD architecture applicable to systems managed by a smart home gateway (SHG). Specifically, the proposed approach aims to reach three goals. The first one is to extend the MUD concept, including the possibility of managing whole smart homes handled by SHGs. The second goal is to protect SHGs and their plug-ins with the MUD standard. The third purpose is to protect every kind of plug-in. I.e., in this solution, MUD is not protecting physical devices but plug-ins (regardless of the functionality they offer).

To achieve these goals, we offer developers the opportunity of declaring the network resources required by their plug-ins. When the integrating objects are not equipped with MUD files, developers can provide MUD specifications instead of the device's manufacturer. Indeed, thanks to this solution and the developers' intervention, even devices that are not natively "MUD-enabled", if integrated into the SHG through a proper plug-in, could be protected via the MUD standard.

To process these requirements, we developed a dedicated plug-in for a widespread open-source python-based SHG: Home Assistant [12]. To be part of our proposed solution, each plug-in must specify its requirements in a MUD-compliant way. After merging these requirements, our solution generates a gateway-level MUD file. Then, the generated MUD file is exposed, allowing the network to enforce these policies.

Currently, Home Assistant does not handle the MUD standard at all. Therefore, we realized a HAss plug-in able to create the MUD file starting from a template (provided together with the proposed plug-in itself). In addition, to enforce the MUD policies, the proposed solution requires the cooperation of a MUD manager located in the same network where the HAss instance is deployed. In the developed proof-of-concept, we adopted osMUD [28], an open-source MUD manager designed to run in an OpenWrt router [33].

Preliminary results gathered with a small set of integrations confirm that the network permissions of the HAss gateway and its plug-in are dynamically and automatically enforced according to the developers' declarative MUD policies.

A. Future Work

In Section V, the paper exposes some limitations of the MUD standard and the proposed architecture. Indeed, some solutions to face them are currently under study.

Specifically, concerning the process of blending the policies the developers created, we spot three issues to address in future work. First of all, detecting errors in developers' requirements is crucial. If some requirement is not well-formed, the proposed solution must ignore it (or find a way to fix that requirement). Secondly, future implementation of the proposed proof-of-concept will be able to understand if there are MUD snippets overriding others (conflict detection). In that case, the implemented solution must be able to decide how to manage such a conflict. Lastly, if some policies properly overlap, it could be reasonable to merge them into a unique policy. This operation would optimize the policy enforcement phase.

Another point to consider is the reliability of the MUD snippets. Even if the SHG signs the generated MUD file, there is no guarantee about the snippets' integrity in this preliminary implementation. Considering this issue, we plan to design a more secure way to build the gateway-level MUD file.

REFERENCES

[1] K. Lasse Lueth, M. Hasan, S. Sinha, S. Annaswamy, P. Wegner, F. Bruegge, and M. Kulezak, "State of IoT—spring 2022," IoT Analytics, Tech. Rep., May 2022. [Online]. Available: https://iot-analytics.com/product/state-of-iot-spring-2022/

[2] M. Schiefer, "Smart home definition and security threats," in *2015 Ninth International Conference on IT Security Incident Management & IT Forensics*, 2015, pp. 114–118.

[3] L. Jiang, D.-Y. Liu, and B. Yang, "Smart home research," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, vol. 2. IEEE, Aug 2004, pp. 659–663.

[4] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: an analysis of IoT devices on home networks," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1169–1185. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-deepak

[5] K. Kafle, K. Moran, S. Manandhar, A. Nadkarni, and D. Poshyvanyk, "Security in centralized data store-based home automation platforms: A systematic analysis of nest and hue," *ACM Trans. Cyber-Phys. Syst.*, vol. 5, no. 1, dec 2021.

[6] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[7] E. Lear, R. Droms, and D. Romascanu, "Manufacturer Usage Description Specification," RFC 8520, Mar. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8520

[8] E. U. A. for Cybersecurity, *Good practices for security of IoT: secure software development lifecycle*. European Network and Information Security Agency, nov 2019. [Online]. Available: https://doi.org/10.2824/742784

[9] D. Dodson, D. Montgomery, W. Polk, M. Ranganathan, M. Souppaya, S. Johnson, A. Kadam, C. Pratt, D. Thakore, M. Walker *et al.*, "Securing small-business and home internet of things (IoT) devices: Mitigating network-based attacks using manufacturer usage description (MUD)," National Institute of Standards and Technology, Tech. Rep., 2021. [Online]. Available: https://doi.org/10.6028/NIST.SP.1800-15

[10] J. L. Hernández-Ramos, S. N. Matheu, A. Feraudo, G. Baldini, J. B. Bernabe, P. Yadav, A. Skarmeta, and P. Bellavista, "Defining the behavior of IoT devices through the MUD standard: Review, challenges, and research directions," *IEEE Access*, vol. 9, pp. 126 265–126 285, 2021.

[11] M. Gajewski, J. M. Batalla, G. Mastorakis, and C. X. Mavromoustakis, "A distributed IDS architecture model for smart home systems," *Cluster Computing*, vol. 22, no. 1, pp. 1739–1749, 2019.

[12] Nabu Casa Inc., "Home assistant," 2023, [accessed 05-may-2022]. [Online]. Available: https://www.home-assistant.io/

[13] openHAB Foundation, "openhab," 2023, [accessed 05-may-2022]. [Online]. Available: https://www.openhab.org/

[14] Krellian Ltd., "Webthings," 2023, [accessed 05-may-2022]. [Online]. Available: https://webthings.io/

[15] E. Bertino and N. Islam, "Botnets and internet of things security," *Computer*, vol. 50, no. 2, pp. 76–79, 2017.

[16] F. Corno and L. Mannella, "Security evaluation of arduino projects developed by hobbyist IoT programmers," *Sensors*, vol. 23, no. 5, 2023.

[17] ——, "A threat model for extensible smart home gateways," in *2022 7th International Conference on Smart and Sustainable Technologies (SpliTech)*. IEEE, July 2022, pp. 1–6.

[18] L. Yang, C. Seasholtz, B. Luo, and F. Li, "Hide your hackable smart home from remote attacks: The multipath onion IoT gateways," in *Computer Security*, J. Lopez, J. Zhou, and M. Soriano, Eds. Cham: Springer International Publishing, 2018, pp. 575–594.

[19] K. A. da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. de Albuquerque, "Internet of things: A survey on machine learning-based intrusion detection approaches," *Computer Networks*, vol. 151, pp. 147–157, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618308739

[20] Z. Jin, Y. M. Lee, C. H. Copass, and Y. Park, "Building system with dynamic manufacturer usage description (MUD) files based on building model queries," aug 2022, uS Patent 11,411,999.

[21] S. M. Sajjad, M. Yousaf, H. Afzal, and M. R. Mufti, "eMUD: Enhanced manufacturer usage description for iot botnets prevention on home wifi routers," *IEEE Access*, vol. 8, pp. 164 200–164 213, 2020.

[22] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format," RFC 8259, Dec. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8259

[23] M. Björklund, "The YANG 1.1 Data Modeling Language," RFC 7950, Aug. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7950

[24] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, Mar. 1997. [Online]. Available: https://www.rfc-editor.org/info/rfc2131

[25] "Ieee standard for local and metropolitan area networks - station and media access control connectivity discovery," *IEEE Std 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009)*, pp. 1–146, March 2016.

[26] IEEE, "Ieee standard for local and metropolitan area networks - secure device identity," *IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009)*, pp. 1–73, Aug 2018.

[27] E. Rescorla, "HTTP Over TLS," RFC 2818, May 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2818

[28] K. Yeich and D. Weller, "Open Source Manufacture Usage Description," 2022, [accessed 10-Oct-2022]. [Online]. Available: https://osmud.org

[29] R. Zalenski, "Firewall technologies," *IEEE potentials*, vol. 21, no. 1, pp. 24–29, Feb 2002.

[30] F. Hu, Q. Hao, and K. Bao, "A survey on Software-Defined Network and OpenFlow: From concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, Fourthquarter 2014.

[31] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, mar 2008. [Online]. Available: https://doi.org/10.1145/1355734.1355746

[32] A. Langiu, C. A. Boano, M. Schuß, and K. Römer, "UpKit: An open-source, portable, and lightweight update framework for constrained IoT devices," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, July 2019, pp. 2101–2112. [Online]. Available: https://doi.org/10.1109/ICDCS.2019.00207

[33] A. Holt and C.-Y. Huang, *OpenWrt*. Cham: Springer International Publishing, 2018, pp. 195–217. [Online]. Available: https://doi.org/10.1007/978-3-319-72977-0_9