This is a pre-print. The original will appear in the proceedings of
3rd Workshop on Internet of Things Security and Privacy (WISP)
held in conjunction with Global IoT Summit 2020, published by IEEE as 978-1-7281-6728-2/20

Towards Privacy-Preserving Local Monitoring and Evaluation of Network Traffic from IoT Devices and Corresponding Mobile Phone Applications

Felix Klement
Chair of IT-Security
University of Passau, Germany
fk@sec.uni-passau.de

Henrich C. Pöhls
PIDS – Passau Institute of Digital Security
University of Passau, Germany
hp@sec.uni-passau.de

Korbinian Spielvogel
Chair of IT-Security
University of Passau, Germany
ks2@sec.uni-passau.de

***Abstract*—This paper describes ways for users to gain an insight into the actual communication flow of their Internet of Things (IoT) devices. The paper's main objective is to enable a comparison of the flow with the devices' intended purpose understandable to the device user. On the basis of what the device sends, the user should be enabled to decide whether the traffic is legitimate or not. With our framework no additional data will leave the user's premises at any time. Only when a user decides that the traffic is *unwanted communication flows* the user can voluntarily transfer selected excerpts to a third party for further analysis. This limits data leakage compared to existing security incident event management (SIEM) solutions, where the monitoring third party seeks to constantly collect all information about the user's traffic and thus constantly gets sensitive information. In this paper we propose a first set of tools for purely local analysis and user-friendly local visualizations. By this we educate the local user/operator of the IoT deployment and allow for more informed and more transparent decisions. Thus, we show that a privacy-preserving and thus more data-protection (GDPR) compliant monitoring of IoT-related network traffic is possible – and showcase how it will look.**

***Keywords*-Internet of Things, monitoring, SIEM, privacy**

I. Introduction

Nowadays the Internet of Things (IoT) is almost on everyone's lips because it surely makes people's lives easier. A large number of devices are sold to and operated by regular end-customers, e.g. smart locks that allow to unlock the front door without having to be physically present [1], [2]. But the IoT has not only found its way into private households. Many applications have been deployed in areas such as smart cities [3], health surveillance [4], agricultural monitoring, etc. [5]. The downside is, that devices for which security and privacy are not built in by-design [6]–[8] are prone [9] to being attacked: Kumar et al. showed that a significant fraction of devices have only weak passwords or use standard HTTP administration passwords left unchanged by most users [10]. Those devices, when operated by unaware non-security-expert users will soon be used by malicious actors for local attacks or as part of bot-nets for networked attacks.

Authors were partially funded by the European Union's H2020 grant n^o780315 (SEMIoTICS). This paper reflects only the authors' views.

Monitoring devices' activity helps, like in the collection of data used in *Security Information and Event Management* (SIEM) systems, but it can not be installed especially by the non-expert users. Moreover, monitoring like in SIEM systems requires large amounts of data about security related events like what network traffic occurs or what action triggered the event. However, only few works try to integrate privacy in the general SIEM for forensics and threat analyses [11], [12], and this challenge has not yet been tackled for the IoT domain. Moreover, the collection of even more data in the IoT domain gives the user, the feeling of living in a "glass house" [13].

Most users of IoT devices have little to no knowledge of their internals. Nevertheless, nobody wants to reveal private data, e.g. about the household or the family. Everyone wants to protect this very sensitive area of privacy and therefore it is essential to educate users and allow for a safer usage. This highlights the paramount importance to support user in locally finding out about the points of weakened security and privacy in their own local deployment and to be enabled to at least get a feeling what these devices are sending outside.

A. Contributions and Outline

In this paper we describe the steps to interconnect technical tools such that existing technology gives users an insight into the actual communication flow of their IoT devices; starting as early as the configuration of new IoT devices to overseeing the daily actual usage of the devices. We present our prototype and base on it our proposal for a framework that supports IoT device users in determining if there are potential weak points, or whether unexpected and unwanted communication occurs. We stress again that the procedures to obtain the required data are in any case a crude invasion of privacy and security and as such we strongly argue that the data must remain local during analysis and presentation.

First, we present related work in Section II and clarify why we think that it is of paramount important to investigate the locally gathered traffic of IoT devices also locally, at least at first. Technical details of a potential gathering framework are provided in Section III. In Section IV we briefly explain

that getting this data mostly requires breaching security and privacy locally. Hence, in our framework this data is collected, stored and analyzed only locally. In Section V we briefly showcase the visualization this can offer users. We conclude in Section VI that the framework enables users to assess if some selected portion of the sensitive data that has been collected should be given to a third party for further analysis; in a great number of cases the user can already decide on its own. In all cases this helps to detect IoT devices that infringe the privacy and security of the user.

II. Related Work

Related work supports that it is helpful for a security and privacy analysis to examine exactly what data they are sending. For example, Mengmeng et al. [14] proposed a framework for automating their examination concerning security-related incidents for IoT. Johnsdottir et al. [15] presented an intuitive and easy-to-use IoT network monitor for consumers to demonstrate vulnerabilities of IoT devices that they have in their own homes. Also the existing literature has started showing that the technical possibilities to gather this data can be quite different: Kumar et al. [10] perform an analysis of IoT devices on home network based on a tool that scans the local network and leverages features like detecting the type and manufacturer of the device and then checking for known weak credentials and common vulnerabilities[1]. Instead of scanning, passive monitoring like in Shahid et al. [16] shows that IoT device recognition works based on the observation that IoT appliances usually perform very specific tasks, thus their network behavior is very predictable. They present a machine learning-based approach to recognize the type of IoT devices that are connected to the network by analyzing streams of sent and received packets. Yet, another approach is to sit on the device itself, Yahya et al. [17] introduce a prototype for a lightweight monitoring system for IoT devices with an agent-manager model.

Firstly, mentioned publications stress the importance of being able to dissect and evaluate the network traffic generated by IoT devices. However, this requires being able to inspect it, which is much harder if it is end-to-end encrypted. Especially, only little work has been done so far concerning the analysis with regard to the use of mobile applications for setting up and controlling IoT home devices. In this paper, we will broaden the data collection to include apps on the phone as well.

Secondly, gathering the network usage in various stages of the IoT life cycle by a third party is –yet another– infringement of privacy. Looking at the application of privacy mechanisms to threat analytics, literature on SIEM-like systems declared the challenges of privacy [11], [12], [18]. Other works tried to apply privacy-friendly technologies to intrusion detection systems [19]–[21]. The peculiarities of IoT were not considered in these works, and our local analysis approach, with a very selective sharing directly follows the principle of data minimization and thus reduces the privacy invasiveness.

[1]The commercial tool is called WiFi Inspector by Avast

III. Data Acquisition Methodology and Tools

In this section, we elaborate on the crawler infrastructure used. Our prototype currently only captures traffic that is generated via WiFi or Ethernet. The general data acquisition methods theoretically apply to other communication standards used in IoT, e.g. LoRAWAN, Sigfox, or ZigBee. It would require to implement –what we call– a *measuring instance* for each protocol. In the framework the data generated by each measuring instance is aggregated through a unified API endpoint.

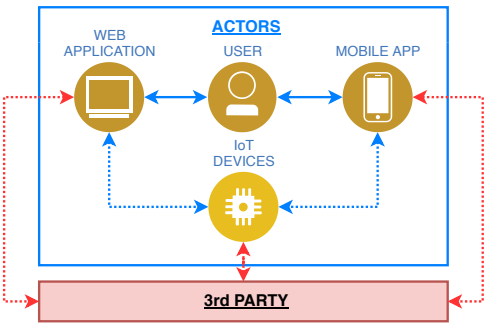

Fig. 1. Interactions of actors in the IoT domain with third parties.

A. Components

In most cases, the so-called Plug'n'Play of IoT devices contains some workflow to integrate the device into a local network. In the majority of cases either via a specific mobile applications or via a web interfaces. For brevity, we focus in this paper on the interactions between IoT devices and possible third parties during and after using mobile apps to set up the specific device. Fig. 1 shows the generic communication of actors in a fictitious IoT network. The dotted lines show the communication exchange which is not necessarily transparent to the user. Within these data flows, the goal is to show what is being sent where and whether the data sent is acceptable traffic.

In addition to the actual actuators, a so-called brainstem is used to collect and evaluate the analysis data. It primarily consists of four components, the *Measuring Instance*, the *Evaluation Instance*, the *Visualization Instance* and the *Storage Unit*. To obtain as much usable data as possible, the measurement component uses a variety of tools to filter the traffic generated in the network and save it in an adapted format. For a complete explanation of each utility see Section III-B. Within the *Evaluation Instance*, the collected data is processed and evaluated. A detailed description of the steps carried out there can be found in Section V. In wanting to present the entire sample optimally for humans, we developed the *Visualization Instance*. A short schematic overview of the brainstem and the general interactions between the single devices can be found in Fig. 2. The *Measuring Instance* consists of several different Python classes. They encapsulate the tools presented in Section III-B, which allows the framework to be modular. The tools inside are responsible for launching and monitoring

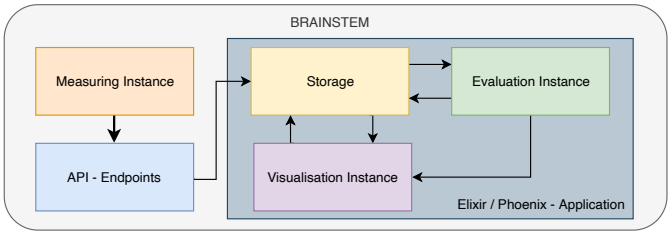

Fig. 2. Interaction overview of the brainstem instances.

the aspect of interest. For example, our *Measuring Instance*s for the IP traffic of IoT devices builds upon the open-source implementation of Princeton's IoT Inspector [22]. The main difference to the approach of the IoT Inspector is that we want to give the user the possibility to get a complete overview of all only-locally collected data himself. This empowers users to carry out own investigations and examinations based on the local data analysis and then, only if deemed ok to share, share selected portions of the data with other users as well.

B. Technical Setup and Tools

The framework consists of three main components. First in our technical prototype, we have a RaspberryPi 3b+ which acts as the brainstem as seen in Fig. 2 and also runs the MITM-Proxy. We operate a wireless nano-router TL-WR802N N300 to intercept the traffic from IoT devices that operate over WiFi. The Pi as well as the wireless router are connected via a standard switch and the user's router to the Internet. Thus, the user's IoT devices and network infrastructure do not need to be modified, especially devices are not isolated behind a firewall. Instead, the Pi is brought into an existing network and still records interesting traffic.

Next, we briefly highlight some selected tools:

1) Detecting Devices via Address Resolution Protocol: The address resolution protocol (ARP) is used for discovering the link-layer address. To explore devices we use the ARP scanning method: We send an ARP-request periodically to all IP addresses in the local subnet and record the responses. New discovered devices are appearing in real-time in the user's web interface. From the interface, users specify explicitly which device should be monitored. To intercept data frames on the network even without being in the privileged position of the router, we use ARP-spoofing. Namely, we emit two ARP spoofing packets every two seconds (similar to Debian's *arpspoof* utility [23]).

One packet is sent to the device to be monitored, using the IP address of the router as the source, and another packet is sent to the router, using the IP address of the device to be monitored as the source. In this way, traffic is redirected, and instead of going from the device to the original router, the packet flows to the Pi. By doing this, we can spy on the communication flow between the device under surveillance and the original router. Since this requires an additional hop on the network, and that packets pass through our monitoring tool chain, the network's latency is slightly increased. Costs of our inspection for N monitored devices can be calculated as follows: If we had a set-up where we wanted to simultaneously monitor 25 devices, we would have *21*((25 + 1)*25)* bytes per second, since each ARP packet typically contains 42 bytes. This sums up to a total bandwidth overhead of 16.7 kilobytes per second.

2) Network packet manipulation: The measuring instance mainly uses the python-based interactive packet manipulation program and a library called *scapy* [24] to parse the captured packets. Here we relied on the existing open-source implementation of the IoT Inspector from Huang et al. [25]. They use the python library to collect pieces of information like the vendor of the network chipset, DNS requests and responses, remote IP addresses/ports and cumulative flow statistics.

3) Network mapping: Netdisco [26] is an open-source library that scans the local network for intelligent home devices using SSDP, mDNS and UPnP. The library is parsing all subsequent responses in JSON strings. These strings may also include the name promoted by a device itself (e.g. "google_home"). An issue with these outputs is that they exhibit a large number of possible variations, like DHCP hostnames. It would be a tremendous technical challenge to develop some sort of regular expression to recognize and validate this data against a set of predefined labels. Another problem with *netdisco* is that some devices do not respond to SSDP, mDNS or UPnP.

4) Port scanning: NMAP, a well-known port scanner, is used for network discovery and is often the basis for security enumeration during the initial stages of a penetration test. *NMAP* displays exposed services on a target machine along with other useful information such as the version and OS by sending packets and analyzing the respective responses. One extremely useful feature is the so-called *NMAP* Scripting Engine (NSE). It allows users to write and distribute simple scripts to automate a variety of network tasks. For convenient execution and triggering different *NMAP* commands or NSE-Scripts from the Phoenix web application we wrote a simple NMAP wrapper for the programming language Elixir called Hades [27].

5) Performing a Man-In-The-Middle attack: The open-source interactive HTTPS proxy *mitmproxy* [28] can be used to intercept, inspect, modify and replay web traffic. We use this tool to intercept the possible encrypted traffic from the mobile application to the manufacturer's server. The MITM in the name represents Man-In-The-Middle – a reference to the process by which we can intercept and disrupt these theoretically opaque data streams. The fundamental concept is to pose as a server to the client and as a client to the server while we sit in the middle and decode traffic from both sides. However, certification authorities are intended to prevent precisely this attack by allowing a trusted third party to cryptographically sign a server's certificates to verify their legitimacy. If this signature comes from an untrusted party or does not match, a secure client will simply terminate the connection. To solve this problem, we use a *mitmproxy* to become a trusted certificate authority ourselves. The tool

includes a full certification authority (CA) implementation that generates and intercept certificates on the fly. To get the client to trust these certificates, we manually register MITM Proxy as a trusted CA on the mobile phone where our app is installed. The scripting API delivers full control over *mitmproxy* and allows you to automatically alter messages, redirect traffic, visualize messages or implement custom commands. This allows us to record and evaluate network traffic in a variety of ways. Currently, we use *mitmdump* to record all network traffic generated between the app and the destination server. A Python script parses the complete recorded content and allows us to store all dumped files in our Postgres database for further investigations.

IV. Invasiveness of Data Gathering

We need a multitude of tools and also rights to be able to gather the needed data. Most of the tools use techniques that can count as direct or indirect network attacks. Even though there are legitimate applications for ARP spoofing such as Linux and BSD-based high availability clusters, it is a typical Man-In-The-Middle attack. If *NMAP* is used improperly, it is also possible that legal limits are exceeded.

There is a wealth of attack limitations where we cannot directly implement our specific data gathering scenario. In general, we assume a completely isolated testbed in which the methods we use do not affect the environment or other devices on the network. The setup would not be practical in production environments, e.g. because of leaked sensitive data or the generated network traffic.

A further restriction concerns the evaluation of the geographical origin of the end servers. If the used IoT device communicates with a server via proxies, the results about the origination of the endpoints are meaningless.

V. Evaluation of Collected Data

This section briefly presents the presentation and processing of the data and the subsequent analysis of the recorded data and explains the results of a trial measurement in more detail. Once you have started a scan for new devices, the list of devices is updated as soon as the framework detects them on the current network. The first time you click on the detail view of a detected device, you will be asked if you want to monitor it or not. In the background, a post request is sent to the python monitoring script to add the selected device to the list of things to monitor. The *Measuring Instance* is configured to only perform the in-depth analysis only for devices in the created whitelist. This step is important, otherwise, every device that is detected would be recorded immediately. When you select a desired device for monitoring, you will be redirected and get an overview of the respective device and the data available at the current time.

A. Data Processing

The procedure for processing the data currently contains four components. These are divided into the processing of the so-called device dicts, the dns dicts, netdisco dicts and the flow dicts. This data processing pipeline can be modularly extended in future researches without further problems. The *Measuring Instance* delivers the recorded data to the *API endpoint* in a *JSON* format. The received data is organized into maps which are more convenient to access in the *Elixir* programming paradigm. New entries in the *PostgreSQL* database are generated for each device not yet known. The monitoring process then saves all data generated by a specific device into the database. Of course, this only happens for devices that have been explicitly selected for monitoring by the user. In order to get information like *remote_ip_owner, remote_ip_city and remote_ip_country* we use the *geolix* library in combination with the free to download *GeoIP2* country and city databases.

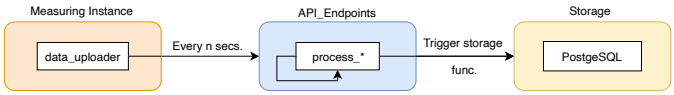

Fig. 3. Data flow during measurement in the brainstem.

The *Measuring Instance* generates a detailed logging file inside its execution folder, which can be very useful for tracking down problems that occur during execution. The system logs a multitude of actions e.g. when uploading data, ARP spoofing results, HTTP responses of the API.

B. Data Analysis Methodology

We use qualitative and quantitative data analysis:

1) Qualitative Data Analysis: One important tool in our qualitative evaluation is the use of the NSE scripts that we already mentioned in Section III-B4. Currently, our prototype uses only the *vulners* script, which outputs known vulnerabilities and information related to it. The scripts to be executed are defined during the measurement setup, as well as the repetition rate and the sleep time between executions. Settings, of course, dependent on the device being tested. For example, facilitating the *bitcoin-getaddr* script that queried the list of known *bitcoin* nodes does not aid when identifying general behavior of a device. Still, such specific scripts provide helpful insights at later stages, e.g. if the user has a suspicion and wishes to investigate further.

Another qualitative method in our evaluation is the creation of a geographical overview, which highlights the countries where packets come in or leave the corresponding origin. We increase the color's intensity according to the number of requests (the stronger the color, the more requests). This is shown in the web application during the measurement in real-time (see Fig. 4). The request's origin is classified on a scale of concern. For example, requests from China, which are potential unwanted requests, are rated higher than requests from Germany.

2) Quantitative Data Analysis: Currently, we have three different quantitative analysis methods. Based on the flow data sent by the python script we can aggregate and compare the used protocol types. In the administration view of the tool, this is presented as a bar graph. Furthermore, we can read the used ports from the flow data and examine them. Another

Fig. 4. Overview of the target countries for the server locations.

striking point is the display of inbound and outbound traffic over a certain time period. Here we use another bar graph, which compares the two recordings at a specific timestamp. This empowers the user to make assumptions about how much data is coming in or going out in relation to the time.

After the specified analysis interval, a detailed report is automatically generated and sent to the inspector. In this technical sheet, we have aggregated all recorded pieces of information explained above. We will expand the scope of the information in the future and try to present the aggregated results even better.

C. Results

To highlight the results obtained by the current implementation, we took the analysis performed on the *YEELIGHT Smart LED Bulb E27 Color*. We started a measurement over six hours and this already showcases what should be possible with the framework even if not yet fully extended. We have created two graphs which can be seen in Fig. 5. Fig. 5a, on the left, represents the number of incoming bytes per measurement for the device that was at rest. The graph on the right, Fig. 5b, shows the number of bytes received per measurement interval when using the lamp (e.g. switching on/off, changing color).

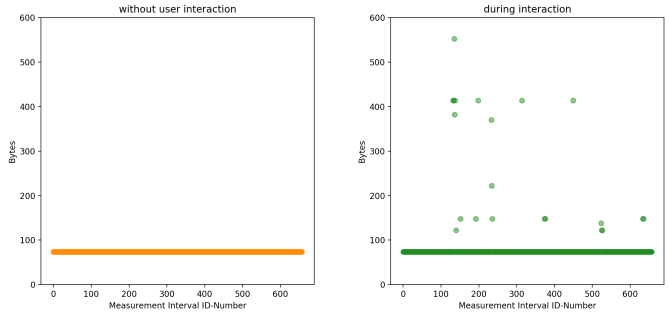

Fig. 5. Inbound bytes of yeelight smart bulb in each interval: (a) left; without user interaction; (b) right; during interaction

The first thing one can deduce from the graph without user interaction on the left (Fig. 5a) is that, network packets arrive despite the resting state of the device, i.e. the user is not interacting with the lamp. We assume that this is a heartbeat to check if the state of the lamp has changed. This could be used e.g. to update the state in the backend to provide the most current state once the user uses the app, or if several users have access to the lamp via different methods. However, in order to make a reliable statement, further research must be carried out by a professional, or more device/protocol/service-specific local analysis *Measuring Instances* need to be provided.

If one compares the two graphs with each other, the graph during user interaction (Fig. 5b) clearly shows some outliers. These are the interactions the user made during the measurement. It is interesting that the number of incoming bytes varies depending on the type of use. This may allow to determine the current usage type based on the value of the bytes received. In this case, measuring a lamp may not be a major intrusion into the private life of a person. However, it can be used to draw conclusions about how often and for how long people stay in certain rooms. Furthermore, there are other internet-enabled devices where you may not necessarily want to be able to trace the way they are used based on the incoming bytes.

D. Future Steps

Our evaluation of the collected data is only the tip of the iceberg, as there is an almost infinite number of ways to evaluate and visualize the collected data. We can, therefore, imagine many more useful applications based on this framework. A fairly large field of possible application is anomaly detection. When we combine collected data with some anomaly detection method, e.g. K-means clustering or multivariate statistical analysis, we could develop a model to differentiate between the typical activity of the device and possible frauds.

Another possibility for future steps is to optimize the amount of data being captured. The prototype currently records almost all the incoming traffic between the selected device, the app and possibly third parties, which results in several gigabytes of data dumps, depending on the measurement period. Maybe just aggregated numbers and not individual packet contents are enough, and thus maybe less privacy-sensitive data can be logged.

Our central idea is to enable the framework to make the data available to the general public which would then allow to create a freely accessible database with information about different IoT devices and their network activity. We envision a voluntary system, in which the user has become educated by the local analytics functions what data the device sends, and thus what data would be contained in the shared network traces, and that users are then willingly consent to share the data with a third party. In exchange the third party will provide them with a more detailed expert analysis based on the ability of the third-party to gather results from several devices. We hope this would foster an ecosystem similar to that of *Virustotal* [29] where users can upload files to check for malware and receive a free report generated by many commercial malware scanners. In exchange, the potentially malicious files of the users may be shared with malware scanner vendors so that detection mechanisms can be further improved. We see a potential to build such a service that collects potentially malicious, unwanted or –in the most benign case– unknown communication of IoT devices with third parties and offers the user an added value when the user shares his traces with the service. For this purpose, the framework would allow the user to send the relevant data, which resides in the local database, to a service for further analysis, especially

if the local analysis was not deemed sufficient. This export would need an agreed reusable data forma for traces.

VI. Conclusion

In this paper, we are primarily concerned with showcasing the idea of a local-only acquisition, first analysis, and user-friendly representation of the gathered data. Our first measurements to test the functionality already show how much traffic is sent unknowingly. In the example of the smart lamp, it is aggravating how many requests actually leave the device even though there is no interaction with it. This also highlights that even simple traces would leak information if constantly transmitted to third parties. The prototype is intended to become available as an open-source tool as a docker container to enable a large number of users to quickly start a comprehensive collection of traces. Moreover, it is the idea that users after becoming aware can share selected network traces with professionals or semi-professionals, to benefit from swarm-intelligence and big-data analysis.

References

[1] R. Satoskar and A. Mishra, "Smart door lock and lighting system using internet of things," *International Journal of Computer Science and Information Technologies*, 2018.

[2] S. Sennan and P. Srinivasan, "Internet of things based digital lock system," *Journal of Computational and Theoretical Nanoscience*, vol. 15, pp. 2758–2763, 09 2018.

[3] G. Moldovan, E. Z. Tragos, A. Fragkiadakis, H. C. Pöhls, and D. Calvo, "An IoT middleware for enhanced security and privacy: the RERUM approach," in *Proc. of 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS 2016)*. IEEE, 2016, pp. 1–5.

[4] C. Frädrich, H. C. Pöhls, W. Popp, N. Rakotondravony, and K. Samelin, "Integrity and Authenticity Protection with Selective Disclosure Control in the Cloud and IoT," in *Proc. of Information and Communications Security (ICICS 2016)*, ser. LNCS, K.-Y. L. et al., Ed., vol. 9977. Springer, 2016.

[5] C. Kolias, A. Stavrou, J. Voas, I. Bojanova, and D. Kuhn, "Learning internet-of-things security "hands-on"," *IEEE Security and Privacy*, vol. 14, pp. 37–46, 01 2016.

[6] N. Petroulakis, E. Lakka, E. Sakic, V. Kulkarni, K. Fysarakis, I. Somarakis, J. Serra, L. Sanabria-Russo, D. Pau, M. Falchetto, D. Presenza, T. Marktscheffel, K. Ramantas, P.-V. Mekikis, L. Ciechomski, and K. Waledzik, "Semiotics architectural framework: End-to-end security, connectivity and interoperability for industrial iot," 06 2019.

[7] A. Kung, F. Kargl, S. Suppan, J. Cuellar, H. C. Pöhls, A. Kapovits, N. Notario, and Y. S. Martin, "A Privacy Engineering Framework for the Internet of Things," in *Proc. of 9th edition of the intl. conference Computers, Privacy and Data Protection 2016 (CDPD 2016)*, January 2016.

[8] H. C. Pöhls, V. Angelakis, S. Suppan, K. Fischer, G. Oikonomou, E. Z. Tragos, R. D. Rodriguez, and T. Mouroutis, "RERUM: Building a Reliable IoT upon Privacy- and Security- enabled Smart Objects," in *Proc. of the IEEE WCNC 2014 Workshop on Internet of Things Communications and Technologies*. IEEE, 2014.

[9] M. Bures, T. Cerny, and B. S. Ahmed, "Internet of things: Current challenges in the quality assurance and testing methods," in *Information Science and Applications 2018*, K. J. Kim and N. Baek, Eds. Singapore: Springer Singapore, 2019, pp. 625–634.

[10] 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)*. USA: USENIX, Aug. 2019, pp. 1169–1185.

[11] M. Jensen, "Challenges of privacy protection in big data analytics," in *2013 IEEE International Congress on Big Data*. IEEE, 2013, pp. 235–238.

[12] P. Stahlberg, G. Miklau, and B. N. Levine, "Threats to privacy in the forensic analysis of database systems," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 91–102.

[13] C. Middleton, "Security: The iot is destroying legal concept of privacy warns in-depth report," https://internetofbusiness.com/glass-houses-the-iot-is-destroying-the-concept-of-privacy-warns-report/, 03 2020, last Accessed: 14.03.2020.

[14] M. Ge, J. B. Hong, W. Guttmann, and D. S. Kim, "A framework for automating security analysis of the internet of things," *J. Netw. Comput. Appl.*, vol. 83, no. C, pp. 12–27, Apr. 2017. [Online]. Available: https://doi.org/10.1016/j.jnca.2017.01.033

[15] G. Jonsdottir, D. Wood, and R. Doshi, "Iot network monitor," in *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, Nov 2017, pp. 1–5.

[16] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "Iot devices recognition through network traffic analysis," 12 2018.

[17] W. Yahya, A. Basuki, P. E. Sakti, and F. F. Fernanda, "Lightweight monitoring system for iot devices," in *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, Oct 2017, pp. 1–4.

[18] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "Sepia: Privacy-preserving aggregation of multi-domain network events and statistics," *Network*, vol. 1, no. 101101, 2010.

[19] M. Sobirey, S. Fischer-Hübner, and K. Rannenberg, "Pseudonymous audit for privacy enhanced intrusion detection," in *Information Security in Research and Business*. Springer, 1997, pp. 151–163.

[20] J. Biskup and U. Flegel, "Transaction-based pseudonyms in audit data for privacy respecting intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2000, pp. 28–48.

[21] H.-A. Park, D. H. Lee, J. Lim, and S. H. Cho, "Ppids: privacy preserving intrusion detection system," in *Pacific-Asia Workshop on Intelligence and Security Informatics*. Springer, 2007, pp. 269–274.

[22] P. I. Inspector. (2020, 03) Iot inspector is a standalone desktop app that lets you analyze your home iot devices. it allows you to determine. Last Accessed: 08.03.2020. [Online]. Available: https://github.com/noise-lab/iot-inspector-client

[23] Debian. (2014, 02) Arpspoof - intercept packets on a switched lan. Last Accessed: 06.03.2020. [Online]. Available: https://manpages.debian.org/jessie/dsniff/arpspoof.8.en.html

[24] Scapy. (2020, 03) Packet crafting for python2 and python3. Last Accessed: 07.03.2020. [Online]. Available: https://scapy.net

[25] D. Y. Huang, N. Apthorpe, G. Acar, F. Li, and N. Feamster, "Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale," 2019.

[26] Netdisco. (2020, 03) Python library to scan local network for services and devices. Last Accessed: 07.03.2020. [Online]. Available: https://github.com/home-assistant/netdisco

[27] Hades. (2020, 03) A wrapper for nmap written in elixir. Last Accessed: 07.03.2020. [Online]. Available: https://github.com/fklement/hades

[28] M. Proxy. (2020, 03) How mitmproxy works. Last Accessed: 07.03.2020. [Online]. Available: https://mitmproxy.readthedocs.io/en/v2.0.2/howmitmproxy.html

[29] Virustotal. (2020, 03) Virustotal homepage. Last Accessed: 14.03.2020. [Online]. Available: https://www.virustotal.com/gui/home/upload

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780315.