Cyber Physical IoT Device Management Using a Lightweight Agent

Matthew Maloney, Elizabeth Reilly, Michael Siegel, Gregory Falco

Working Paper CISL# 2020-02

July 2019

Cybersecurity Interdisciplinary Systems Laboratory (CISL)
Sloan School of Management, Room E62-422
Massachusetts Institute of Technology
Cambridge, MA 02142

Cyber Physical IoT Device Management Using a Lightweight Agent

Matthew Maloney*, Elizabeth Reilly*, Michael Siegel*, Gregory Falco*
*Massachusetts Institute of Technology
{maloneym, reillye, msiegel, gfalco}@mit.edu

***Abstract*—Increasingly IoT-enabled infrastructure such as smart cities, energy delivery systems, communication networks, manufacturing plants and transportation systems are unable to manage devices with various makes, models, configurations and applications. Considering the fragility of these industrial IoT devices and their cyber-physical nature, it is important not to impede on their limited memory and processing power. Existing solutions for IoT device management such as IoT platforms and hardware solutions have considerable downsides. We propose an agent-based mechanism to control and manage diverse IoT devices that has limited impact on device operations. Our agent is tested in the context of delivering security updates to IoT devices and demonstrates the efficiency of the agent-based management architecture.**

I. Introduction

The digitization of critical infrastructure (e.g., smart cities, energy delivery systems) uses cyber-physical internet of things (IoT) devices to manage and control environments. Cyber-physical systems are digital devices that have physical consequences if compromised. Some examples of such infrastructure includes energy delivery systems, water networks and transportation systems. In these environments, IoT devices are used for a wide variety of applications and distributed across concentrated geographical areas. They are produced by a heterogeneous pool of manufacturers, have different operating systems and configurations. These features of IoT present device management challenges for urban infrastructure operators.

IoT device platforms have been heavily marketed by system integrators as the solution for IoT device management. Unfortunately, these platforms tend to be computationally expensive and memory heavy. Also, platforms are generally intended to cultivate an ecosystem of preferred providers for the IoT devices, which locks the operators into certain platform ”partner” original equipment manufacturers (OEMs). The partner OEMs that are part of the platform's ecosystem are not always best-in-class for the intended smart city application or for device security. Should an operating agency want to integrate an IoT device that is outside of the preferred partner ecosystem, additional integration costs are required because a new API will be needed for the device to integrate with the platform. IoT platforms are generally integrated into a predetermined cloud provider, which further limits the flexibility of an operator hoping to manage their IoT assets.

Core functionality for cyber-physical IoT device management revolves around device security and associated updates. Considering the relative fragility of IoT devices, updates must be delivered in a lightweight way that does not interfere with existing processes or potentially brick the device. While pushing over-the-air firmware updates is always an option for device management, it generally requires a system restart which may pose issues for some devices if a malfunction occurs in the process. This is especially important for cyber-physical systems. In 2018, hundreds of traffic lights across NYC went out because of a failed software update [10]. While no injuries were reported, one could imagine vehicle accidents as a result of the disturbance if not managed properly.

We propose a lightweight mechanism to perform security and other device management updates via a golang-based agent. Our agent effectively delivers highly targeted updates to an IoT device's operating system that does not require a system restart to take effect. This reduces the risk for bricking a device or interfering with constantly running processes on devices expected to operate 24/7. Such an operating capacity service level is typical for cyber-physical devices for urban critical infrastructure like an energy delivery system. The agent is capable of providing concurrent process execution, similarly to an IoT platform, with the added benefit of being lightweight and not locking smart cities into a designated ecosystem of devices. Specifically, our agent focuses on securing IoT devices, which is of utmost importance for cyber-physical systems. Many IoT device security solutions are in use today and have been proposed, however they fall short for a variety of reasons outlined below. Our agent aims to systematically address these failures while offering benefits beyond security such as device management for cyber-physical environments.

II. Related Works

IoT device security and management is a persistent issue demonstrated by the many attacks against such systems [16]. Due to the heterogeneity of IoT devices, there are limited mechanisms available to manage a device and its security in a uniform capacity. This has led to a broad range of conceptual and functional solutions that often have niche applicability to certain IoT devices, but are not ubiquitously relevant across the families of IoT. Here we review the merits and challenges with some of the most promising IoT security management techniques.

A. Software/Hardware Hybrid Agent

One approach to IoT device management for security is using the combination of hardware and software to decrease security concerns by increasing the credibility of device data collection [25]. The authors have suggested developing a 'Trustworthy Agent Execution Chip' that would be installed on all devices and would provide a trusted hardware environment on which the software agent can run. The agent is defined as an autonomous piece of software running on the device that could manage resources and regulate actions in order to maximize benefits of the whole IoT system. However, a chip solution requires installation and restart which disrupts the routine behavior of IoT devices. This can be a barrier for manufacturers and can limit the flexibility of cyber-physical system device management.

B. Software Defined Networking/ IoT-IDM

Another mechanism developed for cyber-physical security management is a host based framework for intrusion detection and mitigation that uses popular SDN tools [19]. The authors decided to develop the framework to work within smart home environments at a network level. This decision was to avoid developing embedded software agents for the myriad of devices that may be found within a smart home. This framework addresses network based attacks by managing an inventory of network devices and analyzing traffic. A custom java based module was written to interact with the open SDN controller OpenFlow [2]. This module is responsible for signalling the controller when a network change is needed, either allowing or denying traffic to reach its destination based on the framework detection unit. However, this solution monitors the network as a whole and cannot be simplified for use by specific devices. This again limits the flexibility of cyber-physical device management.

C. Whitelisting in SCADA Sensor Networks

IoT technologies have begun to merge with SCADA networks to more efficiently gather and analyze real time data. This convergence requires an increased level of security management for SCADA devices to avoid compromise. To address this issue, an approach to whitelisting network activity on SCADA networks was developed [14]. The IndusCAP-Gte system works by analyzing a period of regular network traffic on a SCADA network. This traffic is used to build a model for determining abnormal network flows. The analysis phase outputs a set of network rules or whitelists for traffic that is allowed to occur on the network. Enforcement requires the system to be positioned inline between SCADA and field networks and acts upon packets it observes. This solution locks devices into a specific ecosystem in that they must remain inline between SCADA and these field networks. A barrier to such a technique is that existing IoT devices are required to be reconfigured to support this solution. Considering many SCADA systems are legacy devices, this reconfiguration is not necessarily operationaly feasible.

D. Application Whitelisting

Many IoT devices are susceptible to a wide range of attacks and malware [15]. Considering compute power and memory constraints of IoT, traditional anti-virus software applications are not be suitable. Traditional signature anti-virus software also has its own limitations: no protection against zero day attacks, the constant requirement to update signatures, the likelihood of false positives and false negatives, network connectivity requirements for updating signatures and resource constrained devices may not be able to handle and act upon all known signatures [7]. These limitations and the need to defend against malware require a different approach to protect devices. Rather than using a list of known malware signatures, an inverse approach was used, application whitelisting, to only allow known good software signatures to run on devices [7]. In this experiment, the team benchmarked CPU, File IO, and memory utilization of traditional anti-virus versus an application whitelisting approach. They observed the efficacy of using whitelists to protect IoT devices over antivirus when using their benchmarks. Although, this paper was able to show improved CPU metrics, it only tested one antivirus program against a commercial piece of software that enables application whitelisting. The data's sample scope was limited as well considering the only metrics captured were from a Raspberry Pi board. Therefore, it is unclear if this security management solution is scalable to the diverse range of IoT devices.

E. IotProtect: Whitelist-based Protection for IoT Devices

IoTProtect follows a similar but more in depth approach to application whitelisting previously described [7]. The IoT-Protect team also developed and tested an application for the whitelisting of applications on IoT devices [24]. However, this application was designed to be run on a wide range of the IoT devices due to it being written in bash, the default Unix shell for many Linux distributions. This decision avoided many issues with cross compiling agent software to run on different machine architectures. Writing the program in bash kept the program incredibly small at only 1.6Kb, which would allow the program to run on many resource-constrained devices [24]. There were several types of whitelists implemented in the program, pathnames, MD5 hash values of binaries and a command line whitelist. Although the application was small and runs on a wide range of devices, it does have limitations. The checking and detection of IoTProtect only runs periodically, with a default of every sixty seconds. Increasing the check frequency to below thirty seconds started to impact device performance considerably. This time frame may seem small but could still lead to device compromise. The program is also required to be merged into the device kernel or executed in the initial process to avoid being shutdown by malware. The solution was tested against a large set of IoT malware such as the Mirai botnet. The team setup a honey pot, collected samples and labelled them using a third party service Virus Total [4]. While, the IoTProtect showed to be effective in mitigating the impact of malware on a device, it was not an effective preventative mechanism nor can it facilitate security updates.

F. Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles

Manufacturer Usage Description Profiles are an Internet Engineering Task Force proposed standard [18]. The standard hopes to provide a means for devices to signal their networks what type of functionality they require to properly function. By defining what is needed for these devices to operate normally, identifying or defending against unintended network behavior becomes more attainable. MUD was not created to address how networks should authorize requests or to be a substitute for patching and vulnerability management. A MUD profile can provide network administrators additional protection by reducing the threat surface of devices to those intended by the manufacturer. Limiting the threat surface is achieved through access control lists defined in the profile. Due to the nature of IoT devices performing specific functions, many have recognizable communication patterns, building access control lists will be attainable [23]. MUD Profiles already have large industry buy-in with firms like Cisco already contributing to MUD open source projects [1].

Due to MUD Profiles being relatively new and not being an accepted standard for device network behavior profiling, adoption is low. The proposed standard is still useful and will help define how networks should interact with IoT devices. With the possibility of MUD Profiles becoming an accepted standard, there were no existing tools to help manufacturers generate these profiles. To fill this gap, MUDgee was developed to help these manufactures accurately profile their devices network requirements [13, 12]. The system analyzes a devices traffic trace and generates a MUD Profile to spec. This process can generate wrong and invalid profiles if a device is already compromised when the analysis occurs. The process of generating initial profiles was costly from a time perspective, traffic flows were observed over a period of six months for each device. Currently the MUD specification allows both ACCEPT and DROP rules like many firewall implementations but does not consider priority or order of the rules. Authors of the MUDGee tool suggest a whitelist only approach to avoid ambiguity when building MUD Profiles [13, 12]. This idea of network whitelisting mimics application whitelisting, in that it is easier to manage what is intended than to block what is not.

While this solution provides important data about device and network behavior, it does not prevent malware installation nor allow for simple firmware updates. Therefore, MUD is best used as a tactic in IoT security, rather than a comprehensive solution to security device management.

G. Combining MUD Policies with SDN for IoT Intrusion Detection

The ability of MUD Profiles to limit the attack surface of IoT devices is only as effective as the implementation of these policies in the network. Static and dynamic rules can be created based on the MUD profile input. These rules drastically reduce the amount of traffic that is passed forward to the network intrusion detection system (IDS). Each MUD Profile has access control entries (ACEs) that are translated into a set of flow rules [2], these rules are inserted into a switch or router through the SDN controller. While the MUD specification requires rule priorities it does not offer a solution such as only whitelisting [12]. Twenty eight different consumer IoT products were examined and had MUD profiles generated. An analysis of the MUD profiles showed a series of attacks that could be prevented on the devices. A series of volumetric (reflection/amplification, ARP spoofing, port scanning) attacks were launched against four of the original twenty eight devices. These attacks tested the efficacy of using the MUD Profiles in conjunction with an off-the-shelf IDS [3]. An analysis of the launched attacks showed the MUD Profiles limited the attack surface while enabling the IDS to detect attacks by inspecting a small fraction of exception packets. Implementing MUD Profiles makes compromising IoT devices non-trivial and could be used as a foundation for more advanced anomaly or signature based IDS. However, it fails to comprehensively manage IoT device security.

H. IoT Platforms

The explosive growth of network connected devices has led to the development of many IoT cloud platforms. Due to the wide variety of devices and device types, many domain specific platforms have emerged as well. According to a market analysis there are nearly fifty different IoT platforms that currently exist [20]. These platforms cover several domains such as application development, device management, data management and monitoring to name a few. Many of these platforms offer a means for developing and prototyping quickly, while twenty six offer features for specific application domains such as the Industrial IoT [20]. The high number of the different IoT platforms could be out of necessity considering the lack of standardization for how these devices operate could be a reason for IoT platform diversity. The domain-specific nature of these platforms limits their flexibility as different devices would require entirely different platforms. Furthermore, these platforms often require a system restart, which disrupts IoT devices and can impact performance. This is not ideal for cyber-physical device management and can contribute to an ecosystem of required partnerships among certain manufacturers.

I. Industrial IoT Platforms

Industrial IoT devices face different design considerations than most consumer IoT products. Building a IoT platform to service the industrial market must consider, but not be limited to, the following: Energy, how long a device can operate with a limited power supply; Latency, what are the messaging requirements for processing and network propagation; Throughput, how much data can be received and transmitted; Topology, What devices must communicate with each other along with; and Safety and Security concerns [9]. Some of these concerns are not only specific to Industrial IoT but, the concerns are amplified when these systems are built into and are connected to critical infrastructure, as they may cause physical damage and even threaten human lives [22]. Platforms specific to Industrial IoT would suffer from similar drawbacks for IoT device management as traditional IoT platforms.

J. IoT Botnets

A common theme among IoT devices is their susceptibility to hostile takeover via botnet. There are many reasons why botnets have been so effective at overtaking IoT devices [17]. One of the most destructive recent IoT botnets was the Mirai botnet, the authors of which open sourced the code for the bot and control server. Due to the availability of the code, multiple successful variants of Mirai have been released and detected in the wild, using similar attack vectors but targeting different device types [17]. The effectiveness of such a command and control system for IoT devices should be appreciated when conceiving a mechanism for disseminating updates to disparate devices.

III. Architecture

Learning from the challenges faced by the previously discussed material concerning IoT device security management, we chose to base our design on a botnet. Our lightweight agent is only 1.4MB in size and interacts with an IoT device's operating system without interfering with other device processes. Also, we developed features of the agents including process whitelisting (based on the deterministic nature of applications) and network whitelisting (based on the defined behavior of network traffic for cyber-physical systems like energy delivery systems). Further, the agent was strategically programmed in golang.

A. Process Whitelisting

As illustrated in Figure 1, the agent makes use of the Linux /proc filesystem for process and application monitoring. The proc filesystem distills all running processes into directories such as '/proc/ProcID/', where ProcID is the ID of the process running in the OS. Under these directories you can retrieve information on a process such as: executable name, environment variables and command line arguments. The agent utilizes this information to make whitelisting decisions. When the agent is running, it periodically checks the OS to see what is currently running. Its whitelisting decisions are based on a local json file, or one ingested from the central command server. This file contains a whitelist of processes and applications that are allowed to be running on the device. Once ingested the agent compares current running processes to the whitelist. Any discrepancies between what is running and that list is killed by the agent process.

B. Network Filter/Whitelisting

In addition to monitoring applications and processes on device, the agent is also capable of network filtering. Similarly to how applications are whitelisted, IP addresses and domains can also be whitelisted by adding them to the whitelist json configuration file. For network filtering, devices must have support for the Linux kernel module iptables. The module allows for the filtering of network packets by creating ALLOW/DENY rules that match characteristics of the packets such as source, destination IPs and port numbers. A limitation to using the iptables module is, rule order matters; a packet may technically match two existing rules but only the first rule would apply. For this reason a general whitelist approach should be taken, define a default drop policy and explicitly defining ALLOW rules thereafter.

C. GoLang

The golang language was used because it has broad support for cross compiling so that it could be run on a wide range of devices without much investment. Further, as a compiled language, Golang runs natively and efficiently on the devices tested.

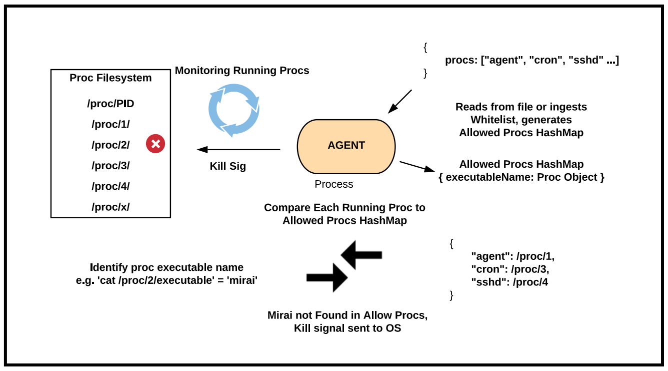
Proc Filesystem
/proc/PID
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/x/
Monitoring Running Procs
Kill Sig
AGENT
Process
{ procs: ["agent", "cron", "sshd" ...] }
Reads from file or ingests Whitelist, generates Allowed Procs HashMap
Allowed Procs HashMap { executableName: Proc Object }
Compare Each Running Proc to Allowed Procs HashMap
Identify proc executable name e.g. 'cat /proc/2/executable' = 'mirai'
{ "agent": /proc/1, "cron": /proc/3, "sshd": /proc/4 }
Mirai not Found in Allow Procs, Kill signal sent to OS

Fig. 1. Agent Process Monitoring

IV. Performance and Metrics

To test the efficacy of the lightweight agent that was developed, several testing scenarios were created. The tests were designed to assess how effectively the agent could handle multiple whitelisting violations.

A. Environment

The performance tests were run on two separate devices: A GL-AR750 MIPS 32bit router running OpenWrT and a GL-B1300 ARM7 64bit router running OpenWrT [5, 6]. ARM and MIPS are two very common chipsets used in IoT devices [8]. A simple bash script was created for testing. This script spawns 'x' amount of new processes depending on number passed as a parameter. The creation of multiple processes simulates a variable compute environment, the more launched the less available memory and CPU available for the agent to run. Each process is launched with the Unix 'time' utility, this is to capture runtime statistics when it closes.

B. Testing

For each device, the test script was run ten times for each of the following scenarios: 5, 10, 25 and 50 processes being generated. The processes that were launched by the script were not included in the process whitelist. When the non-whitelisted process was killed, the time utility prints how long that process ran for. If several processes were launched, the "last time" printout accounts for how long it took the agent to handle all violations. These times were recorded in Figure 2. In Figure 2, the frequency which the agent evaluates its running processes against the whitelist happens every second.

C. Results

The agent performed well under each load's stress test. The agent was able to identify all running processes and shut-down fifty non-whitelisted processes in .4 and 1.13 seconds respectively for the ARM and MIPS routers. The stress testing environment is unlikely to be encountered in field deployed devices but, shows the agent can operate at a high level without negatively impacting device performance. Looking at all cases, the agent continued to perform as one might expect. As the amount of processes launched increased, on average the amount of time required to shut them down also increased. The agent was able to perform more quickly on the ARM router, this is likely attributed to the 64bit architecture and also have more on device memory. The 32bit MIPS router was able to operate using a smaller memory footprint which would be beneficial for heavily constrained IoT devices.

There are several factors that likely skewed the results, making them slightly slower as well. The first factor impacting agent performance was the was profiling library embedded in the agent software. The profile library runs periodically during program execution and was used to calculate the amount of memory the agent consumed while operating (Figure 3). The second factor that impacted agent performance was the test script. The script generates up to 50 additional processes while the agent is running on the device, each of which consumes memory on the device. A future test case may consider starting the processes before the agent is running to avoid using device resources when instantiating the test.

One Second Refresh Window (seconds)												
Procs	Device	1	2	3	4	5	6	7	8	9	10	Average
5	ARM	0.18	0.55	1.21	0.65	1.22	0.16	0.46	0.16	0.87	1.14	0.66
10	ARM	1.27	0.75	1.03	0.49	0.86	0.59	0.28	0.75	0.25	0.79	0.706
25	ARM	0.83	0.48	0.57	1.49	1.02	0.84	0.76	0.42	1.06	1.05	0.852
50	ARM	0.78	0.73	0.73	0.4	0.56	1	1.67	1.67	0.56	1.09	0.919
5	MIPS	0.57	0.72	1.16	0.35	0.93	0.17	0.75	0.32	1.27	0.75	0.699
10	MIPS	0.62	1.39	1.12	0.26	1.02	0.81	0.51	0.61	0.81	0.91	0.806
25	MIPS	0.96	2.11	0.57	0.76	1.15	1.5	1.27	0.63	1.16	0.69	1.08
50	MIPS	1.41	1.37	1.29	1.5	1.59	1.59	1.17	1.13	1.23	1.61	1.389

Fig. 2

Memory Consumption and 1 Second Window	
ARM	175.26kB
MIPS	109.85kB
% Delta	-45.88%

Fig. 3

V. Discussion

As IoT continues to be used for a wide variety of applications and are being produced by disparate manufacturers, we will need a consistent and seamless way to interact, manage and secure these devices. The sensitivity of IoT to memory and processing requirements makes this challenge difficult and for many devices renders heavy solutions such as IoT platforms unfeasible. As demonstrated in our testing, an agent-based management solution for IoT can help achieve the needs of device security management with minimal impact on system efficiency. While our testing focuses on implementing various security features for IoT devices, this is one of many examples of how an agent can be used to manipulate and manage remote devices.

Not dissimilar to how a botnet controls its army of devices, an agent-based mechanism for IoT device management can be highly effective. We believe that the agent proposed can scale similarly to botnets, suggesting that thousands of devices could be under management at a given time by a single control point. Our agent can be seen as a "friendly botnet", which has been demonstrated to be valuable in securing IoT devices in previous studies [11].

Looking ahead, we suggest manufacturers include agent software sockets for each IoT device. This would entail an SDK-like functionality that enables the development community or a purchaser of a series of cyber-physical devices (such as an energy delivery system operator) to develop their own agent that could be used to manage disparate IoT systems remotely. At this time, it is unrealistic to believe that the market will converge on a single IoT device manufacturer so operators of IoT must become comfortable managing IoT devices that were not built to inter-operate. This may be especially true for applications such as energy delivery systems where legacy devices are prevalent and must be used alongside new technology.

VI. Conclusion and Future Work

Our agent solution is a lightweight mechanism that allows for device management and security without compromising the integrity of cyber-physical systems. While many other solutions require system restarts or the integration of hardware components, our agent is able to provide concurrent process execution without considerable impact on the processor or memory. This gives smart city device operators flexibility and peace of mind while updating even the most fragile IoT device. As part of our future work, we hope to integrate the agent with an integrity-first communication protocol like a blockchain-based light client [21]. In combination with this communication protocol, our agent will be able to manage device security updates while guaranteeing communication integrity.

Acknowledgements

Acknowledgments: This material is based in part on work supported by the Department of Energy under Award Number DE-OE0000780.

Disclaimer

References

[1] *MUD-Manager*, (accessed February, 2019). https://github.com/CiscoDevNet/MUD-Manager.

[2] *Open Networking Foundation*, (accessed February, 2019). https://www.opennetworking.org/.

[3] *Snort IDS*, (accessed February, 2019). https://www.snort.org/.

[4] *Virus Total*, (accessed February, 2019). https://www.virustotal.com/.

[5] *glinet*, (accessed March, 2019). https://www.gl-inet.com/products/.

[6] *openwrt*, (accessed March, 2019). https://openwrt.org/.

[7] Raghu Nallani Chakravartula and V Naga Lakshmi. Combating malware with whitelisting in iot-based medical devices. *Int J Comput Appl*, 167(8):33–37, 2017.

[8] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *NDSS*, pages 1–16, 2016.

[9] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.

[10] Erin Durkin. Software malfunction causes hundreds of traffic lights across city to fail. NY Daily News, 2018.

[11] Gregory Falco, Caleb Li, Pavel Fedorov, Carlos Caldera, Rahul Arora, and Kelly Jackson. Neuromesh: Iot security enabled by a blockchain powered botnet vaccine. In *COINS: International Conference on Omni-layer Intelligent systems*, 2019.

[12] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Theophilus A Benson, Matthew Roughan, and Vijay Sivaraman. Verifying and monitoring iots network behavior using mud profiles. *arXiv preprint arXiv:1902.02484*, 2019.

[13] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Matthew Roughan, and Vijay Sivaraman. Clear as mud: Generating, validating and applying iot behavioral profiles. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, pages 8–14. ACM, 2018.

[14] DongHo Kang, ByoungKoo Kim, JungChan Na, and KyoungSon Jhang. Whitelists based multiple filtering techniques in scada sensor networks. *Journal of Applied Mathematics*, 2014, 2014.

[15] Minhaj Ahmad Khan and Khaled Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[16] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[17] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[18] Eliot Lear, Ralph Droms, and Dan Romascanu. Manufacturer usage description specification (work in progress). *Working Draft, IETF Secretariat, Internet-Draft draft-ietf-opsawg-mud-25*, 2018.

[19] Nobakht Medhi, Sivaraman Vijay, and Boreli Roksana. A host-based intrustion detection and mitigation frameowrk for smart home iot using openflow. *11th International Conference on Availability, Reliability and Security*, 2016.

[20] Partha Pratim Ray. A survey of iot cloud platforms. *Future Computing and Informatics Journal*, 1(1-2):35–46, 2016.

[21] Elizabeth Reilly, Matthew Maloney, Michael Siegel, and Gregory Falco. A smart city iot integrity-first communication protocol via an ethereum blockchain light client. *Software Engineering Research and Practices for the Internet of Things*, 2019.

[22] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. Security and privacy challenges in industrial internet of things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[23] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Characterizing and classifying iot traffic in smart cities and campuses. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 559–564. IEEE, 2017.

[24] Chun-Jung Wu, Ying Tie, Satoshi Hara, Kazuki Tamiya, Akira Fujita, Katsunari Yoshioka, and Tsutomu Matsumoto. Iotprotect: Highly deployable whitelist-based protection for low-cost internet-of-things devices. *Journal of Information Processing*, 26:662–672, 2018.

[25] Xu X., Bessis N., and Cao J. An autonomic agent trust model for iot systems. *The 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks*, 2013.