Cyber-Physical System Security Automation through Blockchain Remediation and Execution (SABRE)

Matthew Maloney, Gregory Falco, and Michael Siegel

Working Paper CISL# 2020-12

January 2020

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

Cyber-Physical System Security Automation through Blockchain Remediation and Execution (SABRE)

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

***Abstract*—There are considerable challenges that surround the security of cyber-physical systems. These challenges are compounded by the often heterogeneous nature of different IT and internet of things (IoT) systems that can be found in them. Some of the most onerous tasks around securing a cyber-physical system stem from operational security issues, like patch and update management. Many operational security tasks can be repetitive and prone to error. Managing the security of these systems requires a new approach, one designed to help reduce repetition and tackle common operational security tasks. The Security Automation through Blockchain Remediation and Execution (SABRE) agent was designed specifically to deal with these types of challenges. The SABRE agent aims to reduce complexity and increase the security within a fleet of devices in a cyber-physical system. The solution was built on top of the Etheruem network and designed to operate on large scale cyber-physical systems.**

I. Introduction

Cyber-physical, autonomous systems that are deployed at scale, as is the case for energy delivery systems, face unique security challenges as compared to traditional IT systems. Their direct impacts on the physical world result in major safety implications should the system not function as intended - either by way of an innocent software bug or a malicious attack. Further, their autonomous nature by means of artificial intelligence could perpetuate a given security problem. This necessitates a secure and integral process for configuring cyber-physical systems to assure that the devices are performing in a safe and reliable way.

Cyber-physical systems could be as complex as autonomous transportation or as accessible and thoroughly deployed as co-generation energy plants. Regardless of the use-case, IoT devices deployed in a cyber-physical environment must undergo a configuration process to ensure interoperability with other devices in the system. Device configuration is also critical to ensure security parameters of a device are implemented, a process we call security provisioning. For the purposes of this study, we investigated a co-generation plant's process for implementing security across devices in the plant's distributed control system (DCS) to evaluate how to improve existing operational security efforts.

We learned that new IoT devices such as variable field drives (VFDs), remote terminal units (RTUs), firewalls or programmable logic controllers (PLCs) are scheduled for installation to the DCS environment on a continuous basis, as scheduled as part of an ongoing plant commissioning process. As a result, operators are constantly performing operational system integration procedures to ensure these new devices perform as intended. These devices all have default firmware installed from the manufacturer, which must be configured to the requirements of the plant. Not only does firmware need to be configured, but security provisioning of such devices must be performed as well. The nature of operating a co-generation plant involves constant issue remediation and "fire-fighting", therefore some of these security configurations are not performed when intended, which creates potential opportunities for adversaries to compromise plant operations.

The consistency of what needs to be checked while configuring the security of such IoT devices is an opportunity for automation. Automating security configuration would require a means for reliably detecting potential known issues and remediating them through a reliable process that does not require the time of a busy plant operator. In light of the operational security challenges in cyber-physical systems like our co-generation plant, we have developed a mechanism for detecting security configuration issues, sourcing fixes and implementing security updates to IoT devices. We achieved this using a light-weight blockchain client and associated agent.

II. Operational Security Challenges

IoT device security and management is a persistent issue demonstrated by the many attacks against such systems [1]. 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. The challenges are broad and often changing dependent on the environment in which the IoT device is deployed. Operational security issues such as provisioning or bootstrapping and maintenance are often compounded by the heterogeneity in a fleet of devices. On top of maintenance, properly configuring and segmenting these often insecure devices [2] can be a daunting challenge.

The importance of establishing a security domain for your IoT fleet cannot be overstated [3]. Managing the identities and communications of devices is a critical aspect of security

operations. Being able to ensure identity and attestation on your network begins with understanding what is supposed to be on your network. Establishing identity and secure communications for many IoT devices continues to be a challenge though [4, 5]. The problem is exacerbated by the environment in which these devices are deployed in and the data they collect. Many of these devices are processing mission critical and or private data, prospective goldmines for would be attackers to take advantage of [6]. Establishing secure communications and identity is a solved problem in the larger IT landscape.

Regularly updating and patching systems is one of the best things an organization can do to improve their cyber-hygiene. This is also one of the most challenging operational security tasks an organization can face, which is amplified by IoT deployments. Many devices become less secure over time due to patching and update policies, although it impossible to predict what threats may emerge in the future, over time more vulnerabilities are discovered. For instance, last year nearly 200 million devices were found to be vulnerable to several bugs discovered in a version of VxWorks (the real time operating system) that was released over a decade ago [7]. Patching and updating regularly can reduce an organizations threat profile and reduce the likelihood of a breach [8], yet, many organizations are still struggling to manage this process effectively.

Lastly, ongoing network policy maintenance is a challenge for many organizations. With the introduction of the IoT, many once static environments have to manage an ever increasing amount of devices. When considering network security, the whole network must be secure, going beyond just the perimeter [9]. The network is only as secure as its weakest link, device or communication channel. Deploying IoT can drastically change ones threat profile, increasing ones susceptibility to attack if misconfigured. Creating fine grained policy rules for communications between devices and the perimeter is an ongoing challenge but one that must be managed to ensure secure IoT deployments.

III. Related Works

Here we review the merits and challenges with some of the most promising operational security management techniques for IoT devices.

A. Software/Hardware Hybrid Agent

One approach to operational security challenges in IoT devices is using the combination of hardware and software to decrease security concerns by increasing the credibility of device data collection [10]. These 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 trustworthy chip would ensure the confidentiality of data on the device. 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. Although this approach addresses an operational challenge of ensuring data confidentiality for devices, it introduces a more burdensome operational challenge of replacing or modifying existing devices.

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 [11]. This addresses an operational challenge of securing the network, with a more automated solution. The authors decided to develop their framework to work within 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 system. 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 [12]. 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 solution is also heavily reliant on traffic analysis, which reduces setup costs but may miss some malicious traffic. Although automated, there is no security offered by default without analysis, something that proper segmentation achieves. Basic rules and segmentation of the networks and devices can address many potential low effort security vulnerabilities.

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

Manufacturer Usage Description Profiles are an Internet Engineering Task Force proposed standard [13]. 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. This standard could address the operational security challenge of network management by making normal usage patterns known. Understanding how devices operate would allow for administrators to create and maintain a higher level of granularity in their network policies and definitions. 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 [14]. MUD Profiles already have large industry buy-in with firms like Cisco already contributing to MUD open source projects [15]. An accessible standard will only help to improve operational challenges around network management.

D. 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 [16]. 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. Although this automation helps improve efficiencies and reduce operational security loads, it is reliant on the length of the and accuracy of the analysis phase. 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 operationally feasible.

Blockchain Developments / Device Updates

E. Smart Contracts

A smart contract IoT management system was proposed in [17]. This system proposes using the Ethereum network for the control and configuration of IoT devices. The Ethereum network was proposed for its smart contract capabilities that are built into the network. The authors discuss writing smart contracts to define the behavior of devices such as smart meters. By using smart contracts, the authors assert the data cannot be forged or tampered with by malicious actors.

F. Peer to Peer Firmware Updates via Blockchain

A firmware update scheme that utilizes blockchain to validate firmware versions was proposed in [18]. The blockchain that is proposed would be used for storing hash information of firmware. The downloads and storage of said firmware would occur via a peer to peer network. The proposed scheme enables IoT devices to keep up-to-date with released updates while ensuring integrity and correctness of the firmware. This process of requesting and confirming firmware may cause unnecessary traffic due to the peer to peer nature.

G. Blockchain Firmware/Software Verification System

A system where device software and firmware can be verified against a public blockchain was proposed in [19]. Rather than using the blockchain to distribute device updates, the chain is used to perform validation and integrity checks. The firmware would be stored off chain, but a conceptual 'Reference Integrity Metric'(RIM) would be stored publicly. This RIM would be checked against when an embedded system was updating or installing a new version of firmware. Such a mechanism alleviates some of the traffic concerns as previously described, but may reduce the integrity of the communication because it relies on an external source to the blockchain.

H. Integrity-first Communication for IoT

A secure communication protocol based on the Ethereum Blockchain was built expressly to ensure integral communications across IoT devices [20]. This work aimed to fill a feature gap in many IoT devices, a secure means of transmitting and receiving data. The protocol involved shrinking the Ethereum code-base, allowing it to be run on a wide range of IoT devices, without impacting any integrity benefits inherent to the blockchain.

I. IoT Security Agent

Ensuring communication integrity and providing a means for updates is not sufficient alone to address operational security. An agent must also be leveraged to evaluate the state of the given IoT device in question. A lightweight security agent that can run on a broad range of IoT devices in order to evaluate if certain security precautions are in place has been developed [21]. The agent ingests a whitelist of applications that are allowed to run on the device while terminating other processes. The proposed agent did not have a secure method to receive the whitelist when it was first built.

IV. Architecture

The SABRE architecture entails a command and control (CC) server that operates over the Ethereum blockchain. An agent is deployed on each IoT device that is designed to interact with the CC.

A. Command and Control Server

The command and control servers main responsibility is the security provisioning of deployed devices. This server acts similarly to management servers and parallels can be drawn to some botnet-over-blockchain architectures and defense tools [22]. To fulfill its fleet management requirements the server needs a way to communicate and receive logging information from devices in the field. Communication to the agents is handled through the Ethereum blockchain. The CC can send Ethereum transactions that target an organizations entire fleet of devices, an individual cogen plant, a subset of that plant or an individual device. This is accomplished through the use of semi unique identifiers that can be assigned to the device on startup. This identifier has parallels to a MAC identifier, in that the first characters in the identifier are associated with an organization or company. The next grouping of characters can specify a deployment identifier, that tracks where the device is physically in use, followed by a device type and lastly appended with a unique UID. For example a valid ID structure

will look as follows (CompanyID - SiteID - DeviceTypeID - UniqueID). Scoping of transactions for the agent cascades from left to right and can be specified by any of the groupings.

B. Ethereum Blockchain Implementation

The CC is responsible for publishing Etheruem transactions with the specified security information for an organization. The agent expects certain data and configurations to operate. For instance, the CC is responsible for publishing the most recent firmware version numbers. This information can be published via need-based usage, or as a bundled transaction that contains all configuration data. For example, the CC can publish a whitelist update for devices that have already been provisioned or a bundled transaction that contains information on firmware, whitelists and certificate data. This allows for increased flexibility with regards to device management, only publishing what is needed for operating securely. Figure 1 shows in detail how the CC servers publish transactions with additional whitelist data.

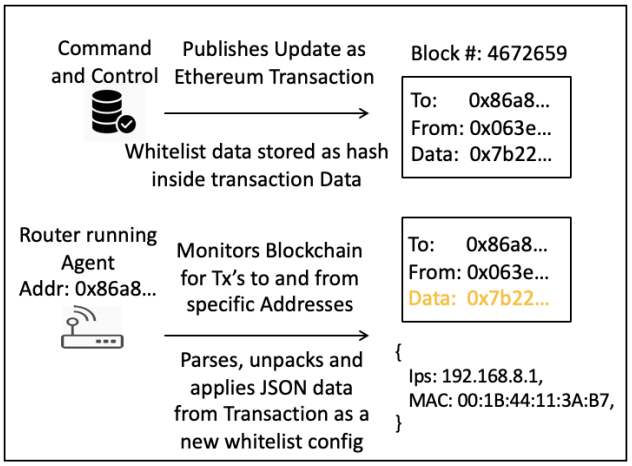

Fig. 1. Ethereum Transaction

To publish transactions, an organization must maintain two Ethereum addresses or wallets, one for sending and one for receiving. The agents are configured to listen to transactions occurring on addresses owned by the organization. The Ethereum protocol allows for additional information to be added to the transaction in the form of a data hash. The CC server encodes the operational security data in base64 format and appends this data into the Ethereum transaction before publishing the transaction to the entire network. The operational security data is held in a JSON object before Base64 encoding occurs. Generally, no Ethereum is actually transacted between the sending and receiving wallet in this exchange but, depending on the amount of data encoded, a certain amount of Ethereum "gas" is spent to publish the transaction. When creating transactions on the Ethereum network, the publisher needs to be cognizant of Ethereum network variables, primarily the gas price and gas limit. Gas price is the cost of doing work on the network and controls the order in which transactions are approved on the network. The higher the price of gas, the higher the cost of publishing a transaction on the network. In addition to there being a gas price per transaction, there is a gas limit. The gas limit is the amount of gas that can be spent for each block of transactions on the network for a given time. Both of these variables are dynamic and change based on real-time network demand. The specific algorithms for determining the gas price and limit can be found in the Ethereum white paper [23]. These two variables directly impact the amount of data that can be published to the network at a given time given budget and network constraints. Traffic across the network at a given time may directly impact the cost of publishing a transaction.

C. GoLang

The Golang language was used because it has broad support for cross compilation. As a compiled language, Golang runs natively and efficiently on the devices tested. The Golang runtime is also compact which allows for deployment across a wide array of environments.

V. SABRE Agent Capabilities

As outlined while describing prevalent operational security challenges, some of the most pressing issues for organizations are managing device updates, segmentation and secure communications. The security automation through blockchain remediation and execution (SABRE) agent was designed to ease these operational security issues for organizations managing many devices. The main features of the SABRE agent includes device firmware checks and updates, network whitelisting and certificate management for secure communications. Each of these features are described below.

A. Firmware Updates

Many IoT devices suffer from vulnerabilities surrounding device updates. Lack of a secure update mechanism is identified as top ten threat from the Open Web Application Security Project's (OWASPs) IoT project [24]. To address this security issue and help organizations manage these updates, we sought to build functionality that facilitated the update process. Receiving firmware updates works similarly to how the agent receives whitelist updates. Yet, there are subtle differences due to the size disparity between network whitelists and the firmware. Depending on the device, firmware may expand beyond the size the of an allowed Ethereum transaction. The size of transaction is controlled by the dynamic network factors of gas price and gas limit.

Considering the constraints, we employed the Ethereum blockchain as a source of truth for firmware, rather than a distribution mechanism. The agent listens to transactions happening on the blockchain, filtering out those that not from the configured CC and addressed to it. Once the agent observes a transaction meant for its device, it begins to unpack and decode the data. The data will contain a SHA256 and a retrieval URL for where the firmware is currently stored. For the proof of concept we used a free online cloud service to temporarily store our device firmware [25]. The SHA256

is placed in the agents memory and a download is begun on the retrieval URL. Once the download is completed, a checksum is run on the file, generating a separate SHA256. A comparison to the SHA256 in memory and the most recently computed SHA256 is completed. If the two values match, the firmware upgrade process begins on the device. This process can take up to several minutes and will restart the device, losing connectivity. If the two values do not match, the download is deleted and a log is sent back to the CC server to alert the operators of the failure. For the proof of concept, the "sysupgrade" OpenBSD utility was used to handle installation of the new firmware. This is a command line utility that simplifies the process of updating a devices firmware.

B. Certificate Management

When provisioning a device or fleet of devices for an organization, an important part is distributing organization keys or certificates to those devices. The distribution of public keys and certificates allows for a secure communication channel and encryption across devices you own. In public key encryption schemes, there needs to be a way to receive the public key of the site, service or device that is being communicated with. This public key is used to sign or encrypt data before it is transmitted and the owner of the public key can decrypt this information using its paired private key. A secure communication channel is then created using public information. The agent and CC server were designed to facilitate the distribution and management of an organizations public keys across its fleet.

Similar to how firmware updates are sent across the Ethereum network, the same can be done with certificates and public keys. The CC can control which devices the keys are intended for being as granular as single device updates. The CC can send out single certificates or chained certificates containing multiple sites or services the device can communicate securely with. When a transaction containing a certificate is received the agent is responsible for verifying and importing the certificate on the device. Once properly installed the device can communicate with servers specified in the certificate securely.

C. Network Filtering/Whitelisting

The agent is capable of filtering network traffic based on certain network and packet qualifiers. Network rules can be written to route, drop or allow traffic based on characteristics such as IP addresses and MAC addresses. They agent can whitelist specific IP and MAC addresses, allowing packets to travel only to known devices and addresses. For network filtering to be enabled, devices must have support for the Linux kernel module iptables. The module is how the agent creates rules for filtering of network packets. The iptables module works by creating ALLOW/DENY rules that match characteristics of the packets such as source, destination IPs, MAC address and port numbers. A limitation to using the iptables module is that 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 define ALLOW rules thereafter.

VI. Testing the Agent

For the proof of concept SABRE agent, we used the Ethereum test network Ropsten [26]. This network was designed to mimic the real Ethereum networks functionality but designed for developers building applications on top of the network. The main advantage to using the Ropsten network is that it is free to use compared to the main Ethereum network. Gas and Ether can be acquired using free 'faucets', or accounts that distribute Ether (without its monetary value) at no cost.

The main goal of testing the SABRE agent was to showcase its functionality. To achieve that goal the agent would need to ingest a variety of the security operations data from the CC server. We sought to engage the agent to receive and dynamically update the configuration of its host device. This principle can equally be applied to receiving public keys or certificates. When a device is being provisioned, it should be able to install and utilize the public key. Lastly, after provisioning a device, we sought to demonstrate a firwmare update on the device facilitated by the agent. The testing was completed using commercial off of the shelf routers. The routers selected came with OpenWRT pre-installed which is an open source router operating system that is easy to work with and test software on. The agent was installed using SCP and run as a daemon on the device.

Testing the provisioning capabilities required us to generate a self signed RSA public key. This key would represent a public key that could be used to communicate securely with a series of services or other devices for an organization. For purposes of our test, we used the openssl library to generate the shared key. Once the key was generated, a test transaction was published on the Ethereum network. This transaction contained the contents of the public key certificate. The agent quickly identified the transaction as valid shortly after the transaction was confirmed by the network. The agent was able to save the public key stored in the transaction. The public transaction can be found and inspected on the public block viewer Etherscan [1].

Similarly, we tested the update capabilities of the agent. The agent was installed on a device running an old version of firmware. The CC published a transaction containing all the requisite information for a secure update: a SHA256, the new version and the URL from which the firmware could be downloaded. Once the transaction was confirmed on the network, the agent ran a version comparison, checking its own version to the version in the transaction. Once the determination that an old version was being utilized, the agent downloaded the updated firmware and ran its own SHA256 of the file. Upon validation of matching hashes, the agent

[1]"https://ropsten.etherscan.io/tx/0x39471ed242cc69da9556d71966d76b83f05b4f0a3baff5947795a45d6f0fca09"

successfully began the update process by utilizing sysupgrade. The transaction can be examined on Etherscan as well [2].

VII. Discussion

Considering the vast scale of future cyber-physical system deployments, there will be many opportunities for incomplete security provisioning that could result in security holes. To mitigate the laborious operational process, SABRE can be used to streamline the security of these devices. Employing an agent to conduct security configuration checks provides a mechanism for ensuring devices are checked consistently, with less opportunity for human error. The blockchain provides a reliable storage location for the necessary security information. The blockchain is both resilient to attack because of its distributed nature and because in order to compromise the integrity of the security information posted to the chain, an attacker would need to achieve a 51% attack on the network, which is difficult for large chains such as Ethereum.

The benefits would primarily be realized for multiple devices that must be provisioned in the same manner. For example, in our co-generation plant, the security consistency of VFDs that are routinely updated could benefit from SABRE considering the large number of VFDs in the plant. The operator would benefit from a work-factor reduction since they would only need to configure the agent once versus ostensibly performing the security provisioning many times for each updated VFD. This benefit would scale accordingly for a wider implementation of SABRE across larger deployments such as an energy delivery system. Equally there are risks to deploying SABRE. Should an operator make an error when configuring SABRE, all devices in scope will be affected, rather than a single device. This could cause system-wide failures; therefore, it will be crucial for only experienced operators to be configuring SABRE.

VIII. Conclusion

Future research will require investigating the operational impacts of employing an agent as described in SABRE - especially for real-time operating systems as are common in autonomous cyber-physical systems. Operational latency or downtime caused by SABRE could limit its usability to certain classes of systems. Additional work is also required to test concurrent security provisioning checks and its impact on memory and processing power of IoT devices. Finally, SABRE should be tested at scale with hundreds of devices in a testbed environment in order to stress-test the provisioning mechanism.

As autonomous, cyber-physical IoT devices continue to be employed for energy delivery systems, we need scalable means for ensuring their security. Conducting proper security provisioning of these devices is critical to assuring the autonomous system functions. SABRE could offer a scalable means for automating the security provisioning of such devices.

[2]"https://ropsten.etherscan.io/tx/0xa70cb53034ff4f15d567a97b90d685648d22a7633815b3e5681c87c42d3b63d7"

Acknowledgements

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

References

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

[2] Mario Ballano Barcena and Candid Wueest. Insecurity in the internet of things. *Security Response, Symantec*, 2015.

[3] Tobias Heer, Oscar Garcia-Morchon, René Hummen, Sye Loong Keoh, Sandeep S Kumar, and Klaus Wehrle. Security challenges in the ip-based internet of things. *Wireless Personal Communications*, 61(3):527–542, 2011.

[4] Aafaf Ouaddah, Hajar Mousannif, Anas Abou Elkalam, and Abdellah Ait Ouahman. Access control in the internet of things: Big challenges and new opportunities. *Computer Networks*, 112:237–262, 2017.

[5] Bruno Bogaz Zarpelão, Rodrigo Sanches Miani, Cláudio Toshio Kawakani, and Sean Carlisto de Alvarenga. A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications*, 84:25–37, 2017.

[6] Mauro Conti, Ali Dehghantanha, Katrin Franke, and Steve Watson. Internet of things security and forensics: Challenges and opportunities, 2018.

[7] *Over 200M devices affected by critical flaws found in real-time operating system*, (accessed January, 2020). https://www.scmagazine.com/home/security-news/vulnerabilities/over-2b-devices-affected-by-critical-flaws-found-in-real\-time-operating-system/.

[8] Bill Brykczynski and Robert A Small. Reducing internet-based intrusions: Effective security patch management. *IEEE software*, 20(1):50–57, 2003.

[9] Bhavya Daya. Network security: History, importance, and future. *University of Florida Department of Electrical and Computer Engineering*, 4, 2013.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] *MUD-Manager*, (accessed January, 2020). https://github.com/CiscoDevNet/MUD-Manager.

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

[17] Seyoung Huh, Sangrae Cho, and Soohyung Kim. Managing iot devices using blockchain platform. In *2017 19th international conference on advanced communication technology (ICACT)*, pages 464–467. IEEE, 2017.

[18] Boohyung Lee and Jong-Hyouk Lee. Blockchain-based secure firmware update for embedded devices in an internet of things environment. *The Journal of Supercomputing*, 73(3):1152–1167, 2017.

[19] Mandrita Banerjee, Junghee Lee, and Kim-Kwang Raymond Choo. A blockchain future for internet of things security: A position paper. *Digital Communications and Networks*, 4(3):149–160, 2018.

[20] Elizabeth Reilly, Matthew Maloney, Michael Siegel, and Gregory Falco. A smart city iot integrity-first communication protocol via an ethereum blockchain light client. In *Proceedings of the International Workshop on Software Engineering Research and Practices for the Internet of Things (SERP4IoT 2019), Marrakech, Morocco*, pages 15–19, 2019.

[21] Matthew Maloney, Elizabeth Reilly, Michael Siegel, and Gregory Falco. Cyber physical iot device management using a lightweight agent. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1009–1014. IEEE, 2019.

[22] Gregory Falco, Caleb Li, Pavel Fedorov, Carlos Caldera, Rahul Arora, and Kelly Jackson. Neuromesh: Iot security enabled by a blockchain powered botnet vaccine. In *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, pages 1–6. ACM, 2019.

[23] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[24] *OWASP Internet of Things Project*, (accessed January, 2020). https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project.

[25] *Transfer.sh Easy file sharing from the command line*, (accessed January, 2020). https://transfer.sh.

[26] *Ropsten testnet PoW Chain*, (accessed January, 2020). https://github.com/ethereum/ropsten.