Program-Aware Fuzzing for MQTT Applications

Luis Gustavo Araujo Rodriguez
Daniel Macêdo Batista
luisgar@ime.usp.br
batista@ime.usp.br
Department of Computer Science - University of São Paulo (USP)
São Paulo, São Paulo, Brazil

ABSTRACT

Over the last few years, MQTT applications have been widely exposed to vulnerabilities because of their weak protocol implementations. For our preliminary research, we conducted background studies to: (1) determine the main cause of vulnerabilities in MQTT applications; and (2) analyze existing MQTT-based testing frameworks. Our preliminary results confirm that MQTT is most susceptible to malformed packets, and its existing testing frameworks are based on blackbox fuzzing, meaning vulnerabilities are difficult and time-consuming to find. Thus, the aim of my research is to study and develop effective fuzzing strategies for the MQTT protocol, thereby contributing to the development of more robust MQTT applications in IoT and Smart Cities.

CCS CONCEPTS

• **Security and privacy** → **Mobile and wireless security**; • **Networks** → **Protocol testing and verification**.

KEYWORDS

MQTT, Internet of Things, Security, Testing, Fuzzing

ACM Reference Format:
Luis Gustavo Araujo Rodriguez and Daniel Macêdo Batista. 2020. Program-Aware Fuzzing for MQTT Applications. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20), July 18–22, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3395363.3402645

1 PROBLEM STATEMENT

The Internet of Things (IoT) is becoming the next step in Internet evolution [22, 24], enabling several types of devices to interact through communication protocols. Among IoT protocols, the Message Queuing Telemetry Transport (MQTT) protocol is considered the best, offering lightweight-messaging and low-bandwidth consumption [4, 5, 16, 30]. Over the last few years, MQTT applications have increased drastically [15, 18]. In fact, MQTT currently ranks as the most popular publish-subscribe protocol [11]. Moreover, MQTT is the most widely-used *IoT protocol* [11], standardized by ISO/IEC 20922 and OASIS.

Currently, research on MQTT security is underdeveloped [15]. In fact, MQTT applications have been widely criticized over the last few years for their lack of security [21] and faulty implementations in real-world environments [2, 3, 7, 18, 28]. Considering this issue, we decided to conduct studies to answer two main background questions: (1) *What are the most common types of flaws in these implementations?*; and (2) *What testing frameworks have been proposed to mitigate these flaws?*

In order to answer the first question, we conducted a manual study to determine the root cause of vulnerabilities. All MQTT-related vulnerabilities (As of April 22nd, 2020) were collected from the National Vulnerability Database (NVD), which is the most widely-used exploit repository [13]. Based on our findings presented in Figure 1, thirty-seven vulnerabilities (71.15%) are triggered by malformed MQTT control packets.

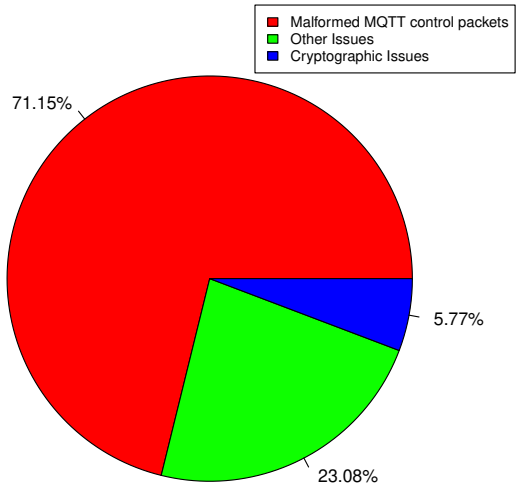

Figure 1: Causes of vulnerabilities in MQTT applications

Malformed control packets could allow information disclosure; remote code execution; and denial of service, thereby hindering confidentiality, integrity, and availability respectively [20]. In fact, a stack overflow vulnerability in MQTT (CVE-2019-11779)[1] was recently discovered by sending a crafted subscribe packet.

Similar to the research by Rodriguez et al. [25], we've analyzed all vulnerability disclosures related to MQTT from NVD. Based on our findings, vulnerabilities triggered by malformed packets

[1] https://nvd.nist.gov/vuln/detail/CVE-2019-11779. Accessed on Apr 30th, 2020

ISSTA '20, July 18–22, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8008-9/20/07…$15.00
https://doi.org/10.1145/3395363.3402645

have higher disclosure delays than their counterparts, hindering immediate measures against cyberattacks. In addition to disclosure delays, deploying and applying patches is usually complex in MQTT environments. For example, open source home automation software such as *HomeAssistant* lack auto-update functionality, meaning MQTT flaws are rarely patched. Moreover, popular home automation devices such as *Sonoff* are vulnerable to malformed MQTT packets [1], and require manual intervention to update as well.

2 RELATED WORK

Testing MQTT applications with effective mechanisms can mitigate these aforementioned issues [7, 27–29]. Thus, in order to answer the second background question (*What testing frameworks have been proposed to mitigate these flaws?*), we conducted a literature review of existing security-testing frameworks for MQTT. Since MQTT is most susceptible to vulnerabilities triggered by malformed packets, *fuzz testing* [19] can play a *key* role in mitigating this issue. In fact, fuzz testing is considered one of the most promising methods for discovering vulnerabilities in IoT [17]. Thus, the following subsections focus specifically on existing fuzzing frameworks for MQTT. The subsections are organized by the year the initial versions of these frameworks were released.

2.1 2015 or Earlier

Defensics[2] is a hybrid blackbox fuzzer, developed by Synopsys, for several protocols, including MQTT. Defensics offers over 250 ready-made test cases; an SDK for custom-made test cases; and a traffic capture fuzzer to generate new test cases. A process monitor and reports are provided for analysis. Although Defensics provides several tools for security-testing, it is not open-source, thereby preventing a higher user adoption.

F-Secure Corporation [12] propose *mqtt-fuzz*, a blackbox mutation-based fuzzer for the MQTT protocol. Its aim is to reduce effort and offer simplicity to developers, thereby lacking complex protocol processing. Although *mqtt-fuzz* is open-source, it requires considerable amount of test cases to reach deep protocol states.

2.2 2017

Anantharaman et al. [3] construct a Finite State Machine (FSM) based on specifications of the MQTT protocol. An input language and parsers were defined and developed for each state respectively. A blackbox generation-based fuzzer was developed to test these parsers. The fuzzer generates test cases based on the input language, meaning it sends correctly-formed packets to the System Under Test (SUT). Rather than using the fuzzer to discover vulnerabilities, this research uses it to test the semantic correctness of the messages.

2.3 2018

Hernández Ramos et al. [15] propose a mutation-based, blackbox fuzzer for the MQTT protocol. The aim of this research is to reduce effort when verifying security of MQTT applications. The architecture of the fuzzer mainly consists of a network sniffer, a template-generator, and the fuzzer itself. The sniffer retrieves network packets between the client and the broker. The template-generator generates and exports templates based on the network packets. From these templates, the user manually marks the fields that will be fuzzed. The fuzzer then mutates these fields by using test cases provided by the user or generating random inputs automatically. The experiments were done with two MQTT implementations: Mosquitto and Moquette. A total of three vulnerabilities were discovered, being mostly denial of service. Although the fuzzer had low CPU consumption, it lacks coverage feedback.

Eclipse Foundation [10] proposes a fuzz testing framework for the MQTT protocol. The fuzzer acts as a man-in-the-middle, mutating network packets between the client and the server. It is a mutation-based blackbox fuzzer, requiring valid templates to generate the test cases. Similar to the fuzzer proposed by Hernández Ramos et al. [15], the user manually selects the messages that will be fuzzed. The developers warn that basic random generators are used to mutate the messages.

Although fuzzing frameworks by Hernández Ramos et al. [15] and the Eclipse Foundation [10] have low-time complexity, they act as a proxy and thus lack support for stateful fuzzing [8, 15].

2.4 2019

Palmieri et al. [21] propose *MQTTSA*, a penetration testing framework for MQTT. *MQTTSA* detects potential vulnerabilities and generates a report offering suggestions to mitigate these issues. *MQTTSA* consists of three penetration-testing mechanisms: authentication bruteforcing, data tampering, and denial of service. Data tampering is based on a mutation-based blackbox fuzzer. Experiments were done with brokers found online and five different deployments of Mosquitto. Approximately 60% of brokers found online lack confidentiality and integrity. Although Mosquitto proves to be secure when configured correctly, this research recommends complementing *MQTTSA* with more effective fuzzing techniques or using F-Secure's fuzzer [12] for better testing.

2.5 Limitations of Existing MQTT Fuzzers

We classified each MQTT fuzzer into three categories (Table 1): understanding the target program; aware of the input structure; and input generation. Analyzing the current state-of-art reveals that existing MQTT fuzzers have several limitations. First, existing MQTT-based testing frameworks are based on blackbox fuzzing, meaning they require considerable amount of test cases to reach deep protocol states [21]. Currently, greybox and whitebox MQTT fuzzers are nonexistent. Second, these blackbox fuzzers discard coverage-increasing inputs during the fuzzing campaign. This means that many inputs could traverse the same path. As a result, vulnerabilities are difficult and time-consuming to find. Third, the probability of false negatives is high because of the lack of program analysis. In addition to the aforementioned fuzzers, popular platforms such as *Peach Fuzzer*[3], *F-Interop*[4], and *American Fuzzy Lop*[5] currently lack support for MQTT. Because of these limitations and lack of support, recent research papers have expressed interest in a *smart MQTT fuzzer* for more effective testing [3, 21].

[2] https://www.synopsys.com/software-integrity/security-testing/fuzz-testing.html. Accessed on November 17th, 2019

[3] https://www.peach.tech/products/peach-fuzzer/. Accessed on August 7th, 2019

[4] https://www.f-interop.eu/. Accessed on June 8th, 2019

[5] https://github.com/google/AFL

Table 1: Classification of existing MQTT fuzzers

Existing Frameworks	Understanding target program			Aware of input structure		Input generation		Open Source
	Blackbox	Greybox	Whitebox	Smart	Naive	Mutation-Based	Generation-Based	
Synopsys	✓			✓		✓	✓	
F-Secure Corporation, 2015 [12]	✓			✓		✓		✓
Anantharaman et al. (2017) [3]	✓			✓			✓	
Hernandez et al. (2018) [15]	✓			✓		✓		✓
Eclipse Foundation, 2018 [10]	✓			✓		✓		✓
Palmieri et al. (2019) [21]	✓			✓		✓		✓

3 PROPOSED RESEARCH

Considering our studies presented in Sections 1 and 2, the aim of the Ph.D. research is to study and develop effective strategies for fuzz testing the MQTT protocol. More specifically, we want to address limitations of existing MQTT fuzzers by using *program analysis*, thereby offering developers a more modern fuzzer than previous works in the literature. Program analysis can be performed through greybox or whitebox approaches. Our goal is to develop a fuzzing framework specifically adapted to MQTT's characteristics.

Analyzing MQTT's vulnerabilities and existing frameworks was crucial for constructing our hypothesis, which is:

> *When using program-aware fuzzing, efficiency is considerably higher than that of existing MQTT fuzzers.*

The research questions that will be used to validate our hypothesis are as follows.

RQ1 How efficient are existing fuzzing frameworks for MQTT?
RQ2 How efficient are program-aware fuzzing techniques for MQTT?
RQ3 What are the characteristics of MQTT vulnerabilities that are revealed by means of fuzz testing?

This research will involve three main efforts: (1) an empirical study of existing MQTT-based testing frameworks; (2) the development of program-aware approaches for fuzzing MQTT applications; and (3) an evaluation of different fuzzing techniques in virtual and real-world environments, determining the most suitable approach for MQTT. The main results expected from this research are: (1) improved understanding of existing MQTT fuzzers; (2) effective strategies for fuzzing MQTT; and (3) robust MQTT applications in IoT and Smart Cities. This research will provide two types of contributions:

Scientific Contribution: The methodology used for this research will support future developments of testing tools either for MQTT, publish-subscribe protocols, or IoT in general.

Technical Contribution: To the best of our knowledge, no greybox or whitebox fuzzer exists for MQTT. Thus, this research will provide the *first program-aware fuzzer for MQTT*. The tools developed for this research will be open-source, allowing developers to test their MQTT applications and improve or build on top of our work for future research.

4 METHODOLOGY AND EVALUATION PLANS

An *MQTT broker* needs to be heavily robust against cyberattacks because it handles requests from publishers and subscribers [6, 14]. Since the broker is considered to be the main component of the publish-subscribe model, this research will focus on testing broker-side implementations of MQTT such as *Mosquitto*.

Our testbed consists of two main components: the fuzzer and the MQTT broker, simulating an attacker and a target system respectively. Both components were configured to communicate with each other on a private network. The goal is to determine the most suitable fuzzing technique for MQTT, whether it be blackbox or program-aware approaches. Test effectiveness or efficiency of each fuzzer (RQ1 and RQ2) will be measured in terms of code coverage; state exploration; unique crashes; and execution times. The following subsections explain how we attempt to tackle each research question.

4.1 RQ1: How efficient are existing fuzzing frameworks for MQTT?

The current state-of-art (Table 1) shows little evidence of the effectiveness of existing fuzzing frameworks for MQTT. Thus, we are currently conducting experiments to evaluate these tools individually. The goal is to analyze their fuzzing efficiency, which will then serve as a baseline to compare with program-aware approaches.

4.1.1 Ongoing Experiments. We are currently conducting experiments with F-Secure's MQTT fuzzer [12] to evaluate its code coverage and vulnerability detection capabilities. Thus far, we tested the most recent versions of Mosquitto (1.6.0 - 1.6.9), using *Gcov* and *AddressSanitizer* as our instrumentation tools.

4.1.2 Preliminary Results. F-Secure's fuzzer injects malformed packets into each version of Mosquitto for twenty-four hours. Approximately ten million packets were exchanged between the fuzzer and the broker during the campaign. However, in most cases (Versions 1.6.0, 1.6.1, 1.6.3, 1.6.4, and 1.6.5 of Mosquito) only 32% of the code was covered by the fuzzer. In three cases (Versions 1.6.2, 1.6.6, and 1.6.8) only 33% of the code was covered and in two cases (Versions 1.6.7 and 1.6.9) only 31% of the code was covered. This means that most test cases traversed through the same path, mainly because of the lack of coverage feedback. Moreover, neither existing nor non-existing vulnerabilities were triggered during the fuzzing campaign. We've also conducted a few experiments with *MQTTSA*'s fuzzer. It managed to cover at most 25% of *Mosquitto*'s source code, further highlighting the inefficiency of public MQTT fuzzers.

We plan to continue performing experiments with other open-source fuzzing frameworks (Table 1) to evaluate their efficiency.

4.2 RQ2: How efficient are program-aware fuzzing techniques for MQTT?

Preliminary results from **RQ1** suggest the necessity of program-aware approaches for fuzzing MQTT applications. With the insights from **RQ1**, we will develop greybox or whitebox approaches that attempt to mitigate limitations of existing frameworks. Since no program-aware fuzzer exists for MQTT, there is a wide range of techniques that can be considered for this research such as guided-fuzzing or symbolic execution.

At the time of writing, we plan to focus first on greybox approaches for improving fuzzing efficiency. Whitebox techniques will be considered depending on our research schedule. Greybox approaches were selected because they combine advantages of both blackbox and whitebox fuzzers, using lightweight instrumentation with minimal overhead. More specifically, we plan on using *Genetic or evolutionary algorithms*, for traversing unexplored or specific regions of the code during the fuzzing campaign. Figure 2 presents an initial flow diagram of each component that will be considered for developing our program-aware fuzzer.

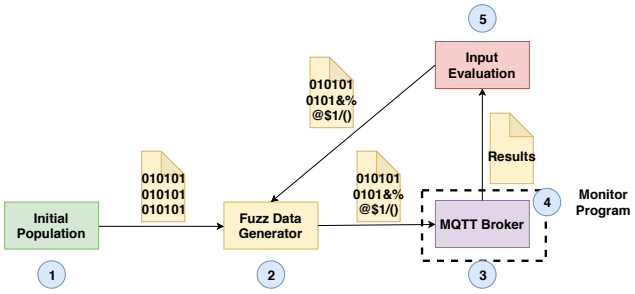

Figure 2: Architecture of a program-aware fuzzer for MQTT

4.2.1 Initial Population. The initial test cases will be generated by capturing valid packets using tools such as Wireshark or Scapy. This approach has been used in several frameworks for MQTT [12, 15]. However, preliminary results from **RQ1** suggest that existing MQTT fuzzers may have difficulties in triggering existing or non-existing vulnerabilities. A more effective approach would be to generate initial test cases based on existing vulnerabilities. Thus, we plan to build a dataset of malicious inputs from existing CVE reports, which will be used as *seeds* to trigger potential flaws.

4.2.2 Fuzz Data Generator. Preliminary results from **RQ1** suggest that existing MQTT fuzzers achieve low state coverage. Thus, our fuzzer will interact directly with the target system, rather than acting as a man-in-the-middle. This approach was chosen in order to offer a more flexible framework, capable of generating test cases based on existing vulnerabilities, a grammar, or whitebox approaches. Best practices for mutating inputs will also be studied.

4.2.3 MQTT Broker. Open-source brokers such as *Mosquitto*; *Moquette*; *Emqttd*; and *RabbitMQ* will be considered to enable program-aware approaches, and thus improve the accuracy of the tests [28]. These brokers were selected because of their popularity and inconsistencies with the protocol standard [9, 15, 20].

4.2.4 Monitor Program and Input Evaluation. Once a test case is selected and injected to the target system, the most effective test cases will be kept for future iterations. Since preliminary results from **RQ1** reveal that most test cases traverse through the same path, the criteria (or *fitness function*) for selecting test cases will be based on code coverage or state exploration. We plan to perform the following activities: (1) study and complement our approach with effective techniques such as stateful fuzzing, which has been successful for network protocols [8, 23]; (2) conduct a qualitative analysis to receive feedback from developers on our fuzzing approach and important metrics to test MQTT applications; (3) use input minimization [26], by reducing and keeping the most effective test cases; (4) select efficient evolutionary algorithms for generating test cases; (5) test MQTT applications deployed in virtual and real-world environments; and (6) analyze experiment results and determine the most effective approach for testing MQTT applications.

4.3 RQ3: What are the characteristics of MQTT vulnerabilities that are revealed by means of fuzz testing?

We plan to analyze the characteristics of vulnerabilities found during our experiments. This analysis could allow us to create a taxonomy to classify security vulnerabilities in MQTT applications.

5 CONCLUSIONS

MQTT is currently being integrated into several IoT-based applications, such as home automation systems; health monitoring systems; and intelligent transportation systems. Considering these scenarios, it is important to mitigate potential attacks and increase reliability of MQTT applications. However, MQTT applications have been widely criticized over the last few years for their weak protocol implementations. Our background studies confirm that MQTT is most susceptible to malformed packets, hence the importance of fuzz testing. Existing MQTT fuzzers are based on blackbox testing, lacking any type of program analysis, thereby offering difficulty to discover vulnerabilities. Thus, the aim of this research is to study and develop effective strategies for fuzz testing the MQTT protocol. We are currently performing experiments with existing frameworks to evaluate their efficiency, which will be crucial when developing program-aware approaches for fuzzing MQTT applications. Our program-aware fuzzer will be open-source, allowing developers to test their MQTT applications effectively, thereby deploying more robust MQTT applications in IoT and Smart Cities.

ACKNOWLEDGMENTS

We would like to thank Professor Maurício Aniche from TU Delft, and Professor Paulo Lício de Geus from UNICAMP for their suggestions to improve our research. This paper would not have been possible without their support. This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9. This research is also funded by FAPESP proc. 18/22979-2 and CAPES.

REFERENCES

[1] Daniel Abeles and Moshe Zioni. 2019. MQTT-PWN Documentation. https://buildmedia.readthedocs.org/media/pdf/mqtt-pwn/latest/mqtt-pwn.pdf. [Online; accessed 17-September-2019].

[2] Khalid Alghamdi, Ali Alqazzaz, Anyi Liu, and Hua Ming. 2018. Iotverif: An automated tool to verify SSL/TLS certificate validation in android MQTT client applications. In *Proceedings of the ACM Conference on Data and Application Security and Privacy*. 95–102. https://doi.org/10.1145/3176258.3176334

[3] P. Anantharaman, M. Locasto, G. F. Ciocarlie, and U. Lindqvist. 2017. Building Hardened Internet-of-Things Clients with Language-Theoretic Security. In *Proceedings of the IEEE Security and Privacy Workshops*. 120–126. https://doi.org/10.1109/SPW.2017.36

[4] S. Andy, B. Rahardjo, and B. Hanindhito. 2017. Attack scenarios and security analysis of MQTT communication protocol in IoT system. In *Proceedings of the International Conference on Electrical Engineering, Computer Science and Informatics*. 1–6. https://doi.org/10.1109/EECSI.2017.8239179

[5] Joseph Jose Anthraper and Joseph Kotak. 2017. Security, Privacy and Forensic Concern of MQTT Protocol. In *Proceedings of the International Conference on Sustainable Computing in Science, Technology and Management*. 1–8. https://doi.org/10.2139/ssrn.3355193

[6] João Antunes, Nuno Neves, Miguel Correia, Paulo Verissimo, and Rui Neves. 2010. Vulnerability discovery with attack injection. *IEEE Transactions on Software Engineering* 36, 3 (2010), 357–370. https://doi.org/10.1109/TSE.2009.91

[7] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, Xiaofeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. 2018. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing. In *Network and Distributed Systems Security Symposium*. https://doi.org/10.14722/ndss.2018.23159

[8] Yurong Chen, Tian lan, and Guru Venkataramani. 2019. Exploring Effective Fuzzing Strategies to Analyze Communication Protocols. In *Proceedings of the 3rd ACM Workshop on Forming an Ecosystem Around Software Transformation* (London, United Kingdom). Association for Computing Machinery, New York, NY, USA, 17–23. https://doi.org/10.1145/3338502.3359762

[9] D. L. de Oliveira, A. F. da S. Veloso, J. V. V. Sobral, R. A. L. Rabêlo, J. J. P. C. Rodrigues, and P. Solic. 2019. Performance Evaluation of MQTT Brokers in the Internet of Things for Smart Cities. In *Proceedings of the International Conference on Smart and Sustainable Technologies*. 1–6. https://doi.org/10.23919/SpliTech.2019.8783166

[10] Eclipse Foundation. 2018. Eclipse IoT-Testware. https://iottestware.readthedocs.io/en/development/smart_fuzzer.html. [Online; accessed 17-October-2019].

[11] Eclipse Foundation. 2019. IoT Developer Survey. https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2019.pdf. [Online; accessed 15-June-2019].

[12] F-Secure Corporation. 2015. A simple fuzzer for the MQTT protocol. https://github.com/F-Secure/mqtt_fuzz. [Online; accessed 16-September-2019].

[13] Ming Fang and Munawar Hafiz. 2014. Discovering Buffer Overflow Vulnerabilities in the Wild: An Empirical Study. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, Article 23, 10 pages. https://doi.org/10.1145/2652524.2652533

[14] S. N. Firdous, Z. Baig, C. Valli, and A. Ibrahim. 2017. Modelling and Evaluation of Malicious Attacks against the IoT MQTT Protocol. In *Proceedings of the IEEE International Conference on Internet of Things and the IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data*. 748–755. https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2017.115

[15] Santiago Hernández Ramos, M. Teresa Villalba, and Raquel Lacuesta. 2018. MQTT Security: A Novel Fuzzing Approach. *Wireless Communications and Mobile Computing* 2018 (2018), 1–11. https://doi.org/10.1155/2018/8261746

[16] Rob Kitchin and Martin Dodge. 2019. The (In)Security of Smart Cities: Vulnerabilities, Risks, Mitigation, and Prevention. *Journal of Urban Technology* 26, 2 (2019). https://doi.org/10.1080/10630732.2017.1408002

[17] Jian-Zhen Luo, Chun Shan, Jun Cai, and Yan Liu. 2018. IoT Application-Layer Protocol Vulnerability Detection using Reverse Engineering. *Symmetry* 10, 11 (2018), 561. https://doi.org/10.3390/sym10110561

[18] Federico Maggi, Rainer Vosseler, and Davide Quarta. 2018. The Fragility of Industrial IoT's Data Backbone. https://documents.trendmicro.com/assets/white_papers/wp-the-fragility-of-industrial-IoTs-data-backbone.pdf

[19] Valentin Jean Marie Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J Schwartz, and Maverick Woo. 2019. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Transactions on Software Engineering* (2019).

[20] Kristiyan Mladenov, Stijn Van Winsen, Chris Mavrakis, and KPMG Cyber. 2017. *Formal verification of the implementation of the MQTT protocol in IoT devices*. Master's dissertation. University of Amsterdam.

[21] Andrea Palmieri, Paolo Prem, Silvio Ranise, Umberto Morelli, and Tahir Ahmad. 2019. MQTTSA : A Tool for Automatically Assisting the Secure Deployments of MQTT brokers. In *IEEE World Congress on Services*, Vol. 2642-939X. IEEE, 47–53. https://doi.org/10.1109/SERVICES.2019.00023

[22] Yusuf Perwej, Majzoob Omer, Osama Sheta, Hani Harb, and Mohammed Adrees. 2019. The Future of Internet of Things (IoT) and Its Empowering Technology. *International Journal of Engineering Science and Computing* Volume 9 (March 2019), 20192 – 20203.

[23] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. 2020. AFLNet: A Greybox Fuzzer for Network Protocols. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation : Testing Tools Track*.

[24] Pedro Martins Pontes, Bruno Lima, and João Pascoal Faria. 2018. Izinto: A Pattern-Based IoT Testing Framework. In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops* (Amsterdam, Netherlands). Association for Computing Machinery, New York, NY, USA, 125–131. https://doi.org/10.1145/3236454.3236511

[25] Luis Gustavo Araujo Rodriguez, Julia Selvatici Trazzi, Victor Fossaluza, Rodrigo Campiolo, and Daniel Macêdo Batista. 2018. Analysis of Vulnerability Disclosure Delays from the National Vulnerability Database. In *Proceedings of the Workshop on CyberSecurity in Connected Devices in the Brazilian Symposium on Computer Networks and Distributed Systems*. https://portaldeconteudo.sbc.org.br/index.php/wscdc/article/view/2394

[26] Gary J Saavedra, Kathryn N Rodhouse, Daniel M Dunlavy, and Philip W Kegelmeyer. 2019. A Review of Machine Learning Applications in Fuzzing. *arXiv* (2019).

[27] M. Schiefer. 2015. Smart Home Definition and Security Threats. In *Proceedings of the International Conference on IT Security Incident Management IT Forensics*. 114–118. https://doi.org/10.1109/IMF.2015.17

[28] Synopsis. 2017. State of Fuzzing. https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/state-of-fuzzing-2017.pdf. [Online; accessed 15-June-2019].

[29] C. Săndescu, O. Grigorescu, R. Rughiniș, R. Deaconescu, and M. Calin. 2018. Why IoT security is failing. The Need of a Test Driven Security Approach. In *Proceedings of the International Conference: Networking in Education and Research*. 1–6. https://doi.org/10.1109/ROEDUNET.2018.8514135

[30] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi. 2017. Internet of Things: Survey and open issues of MQTT protocol. In *Proceedings of the International Conference on Engineering MIS*. 1–6. https://doi.org/10.1109/ICEMIS.2017.8273112