



HAL
open science

A guiding framework for vetting the Internet of things

Fatma Masmoudi, Zakaria Maamar, Mohamed Sellami, Ali Ismail Awad,
Vanilson Burégio

► **To cite this version:**

Fatma Masmoudi, Zakaria Maamar, Mohamed Sellami, Ali Ismail Awad, Vanilson Burégio. A guiding framework for vetting the Internet of things. *Journal of information security and applications*, 2020, 55, pp.102644:1-102644:11. 10.1016/j.jisa.2020.102644 . hal-02984706

HAL Id: hal-02984706

<https://hal.science/hal-02984706v1>

Submitted on 24 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

A Guiding Framework for Vetting the Internet of Things

Fatma Masmoudi^a, Zakaria Maamar^b, Mohamed Sellami^c, Ali Ismail Awad^{d,e,f}, Vanilson Burégio^g

^aCollege of Computer Engineering and Science, Information Systems Department
Prince Sattam Bin Abdulaziz University, Al Kharj, Saudi Arabia

^bCollege of Technological Innovation, Zayed University, Dubai, UAE

^cSamovar, Télécom SudParis, Institut Polytechnique de Paris, Paris, France

^dDepartment of Computer Science, Electrical and Space Engineering, Luleå University of Technology, 97187 Luleå, Sweden

^eElectrical Engineering Department, Faculty of Engineering, Al-Azhar University at Qena, Qena 83513, Egypt

^fCentre for Security, Communications and Network Research, University of Plymouth, Plymouth PL4 8AA, U.K.

^gDepartment of Computing, Federal Rural University of Pernambuco, Recife, Brazil

Abstract

Like any emerging and disruptive technology, multiple obstacles are slowing down the Internet of Things (IoT) expansion for instance, multiplicity of things' standards, users' reluctance and sometimes rejection due to privacy invasion, and limited IoT platform interoperability. IoT expansion is also accompanied by the widespread use of mobile apps supporting anywhere, anytime service provisioning to users. By analogy to vetting mobile apps, this paper addresses the lack of principles and techniques for vetting IoT devices (things) in preparation for their integration into mission-critical systems. Things have got vulnerabilities that should be discovered and assessed through proper device vetting. Unfortunately, this is not happening. Rather than sensing a nuclear turbine's steam level, a thing could collect some sensitive data about the turbine without the knowledge of users and leak these data to third parties. This paper presents a guiding framework that defines the concepts of, principles of, and techniques for thing vetting as a pro-active response to potential things' vulnerabilities.

Keywords: Internet of Things, Security vulnerabilities, Vetting, and Atomic/composite duties.

1. Introduction

In the 21st century, security triad formed by confidentiality, integrity, and availability (CIA) in addition to user privacy, are among many pressing concerns that the Information and Communication Technology (ICT) community is actively examining [1]. The number of misuse and fraudulent cases related to the ICT are on the rise, suggesting the need for an immediate revision of existing practices, techniques, and tools. According to the Australian Institute of Criminology, the estimated value of internal fraud against the Commonwealth has steadily increased from \$1.9 million to \$3 million between

Email address: mohamed.sellami@telecom-sudparis.eu (Mohamed Sellami)

8 2008 and 2011. To mitigate some of these cases, an option is to **vet** ICT applications prior to their in-
9 tegration into day-to-day (sometimes critical) business operations. By vetting, we mean ensuring their
10 safety and compliance with relevant regulations. Mobile apps exemplify ICT applications, whose rapid
11 and uncontrolled widespread use has become a major concern to policy makers. As of September 2019,
12 there were 2.7 million apps posted on *Google Play Store* and 2.46 million apps posted on *Apple's App Store*,
13 which are the 2 leading app stores in the world [2], with the number of connected devices set to top 20
14 billion by 2023 [3].

15 In conjunction with the mobile apps “fever”, we observe some early signs of another “fever” that the
16 Internet of Things (IoT) could end-up catching. According to Gartner, 6.4 billion connected things were
17 in use in 2016, and this number is projected to reach 20.8 billion by 2020 [4], [5]. Accordingly, the total
18 economic impact of IoT will reach between \$3.9 trillion and \$11.1 trillion per year by 2025 [6]. This rapid
19 increase in the number of things needs to be closely monitored to avoid a “boom” in privacy breaches,
20 identity thefts, confidentiality deceptions, etc.

21 Things in an IoT ecosystem have some unique features that make them different from other compo-
22 nents such as reduced size, restricted connectivity, continuous mobility, limited energy, and constrained
23 storage [7]. On top of these features, things’ plethora of uses lead to the generation of rich and large
24 volume of (unstructured) data that need to be processed in compliance with requirements such as safety,
25 security and privacy. The complexity of these features mixed with these different uses raises a number
26 of challenges related to how safe things are or should be. In this context, \mathcal{V} etting IoT (\mathcal{V} IoT) is appro-
27 priate for ensuring things’ “good conducts” before allowing them to collect, process, and distribute data
28 without raising any concerns. However, there is a limited number of R&D initiatives on \mathcal{V} IoT, which is
29 accentuated by the heavy dependence of ICT practitioners on the security claims of things’ vendors [8].

30 By analogy to mobile apps vetting, we aim to develop a guiding framework for \mathcal{V} IoT. The objective
31 is to verify things’ “good conducts” and identify the vulnerabilities that could lead to misconducts.
32 Examples of vulnerabilities could be the excessive use of resources, data sharing with unauthorized
33 parties, and changes in initial configurations. In this paper and in line with our ongoing agenda on
34 IoT [9, 10], we refer to things’ actions as duties and categorize them into atomic and composite ¹. This
35 categorization permits the fine-tuning of the vetting according to each duty’s characteristics and whether
36 the duties originate from the same or different things. Our contribution is manifold, including (i) the

¹Atomic/Composite duty is similar to Component/Composite service adopted by the service computing community [11]

37 definition of vetting in the context of IoT, *(ii)* the identification of vulnerabilities for atomic duties and
38 composite duties, *(iii)* the analysis of the consequences of vulnerabilities on the completion of duties,
39 and *(iv)* the demonstration of the vetting through a proof of concept.

40 The rest of this paper is organized as follows. Section 2 discusses initiatives for vetting mobile apps.
41 Section 3 identifies existing gaps in vetting things. Section 4 details the process of preparing things for
42 vetting. This consists of identifying their duties along with analyzing the impact of vulnerabilities on
43 these duties that things are expected to achieve. Prior to concluding in Section 6, conceptual details
44 about the framework along with the results of the experiments are presented in Section 5.

45 2. Vetting mobile apps

46 According to the US National Institute of Standards and Technology (NIST) [12], there is a need to
47 secure mobile apps from vulnerabilities and defects; apps are used by all people and all organizations.
48 To satisfy this need, a strict vetting process would ensure that mobile apps comply with an organiza-
49 tion's security requirements and are to a certain extent free of (serious) vulnerabilities. These require-
50 ments could be related to origin, data sensitivity, and target environment and could be decomposed
51 into *(i)* general with respect to some existing standards and best practices, such as those specified by
52 NIAP, OWASP, MITRE, and NIST, and *(ii)* specific with respect to some internal policies, regulations,
53 and guidelines. Factors that could be the cause of app vulnerabilities include design flaws and pro-
54 gramming errors, which could have been inserted intentionally or inadvertently [12]. Depending on the
55 risk tolerance of an organization, some vulnerabilities might be more serious than others, suggesting the
56 need for a contextualized vetting process. Vetting could occur throughout an app's lifecycle, which con-
57 sists of development, acquisition, and deployment stages and would cover correctness testing, source
58 and binary code testing, and static and dynamic testing.

59 In [8], Quirolgico et al. mention that millions of apps for mobile devices are available through com-
60 mercial stores and open repositories. Because of their low cost and widespread, the threats of the vulner-
61 abilities of these apps could be far greater than those of traditional computers. Because some vulnerabil-
62 ities of mobile apps are unique, the study insists on the urgency of developing a quick and cost efficient
63 vetting process.

64 In [13], He et al. analyze malware attacks on smartphones and suggest a two-level defense strategy
65 that aims at preventing malware from entering smartphones and determining whether appropriate tools
66 should detect and remove them. However, this may not always be successful due to the rapid changes in

malware behavior and therefore, more sophisticated tools need to be quickly developed while considering smartphones' limitations. He et al. recommend collaboration between apps' administrators, phone users, and application developers to detect and prevent malware threats. They highlight the need for a strict app vetting on the server-side for removing any malicious apps from the market using cloud-based scanning engines. Finally, He et al. conclude by suggesting new research directions related to the issues of (1) how to systematically collect data of emerging malware; (2) how to develop techniques and tools to associate new attacks with those already recorded; (3) how to uncover unknown malware; and (4) how to make a paradigm shift that will involve advertisers as defenders, too. The lack of datasets about new vulnerable applications is also discussed by Nagappan and Shihab in [14]. They address the need for more sophisticated lightweight tools that would prevent malicious code from getting into smartphones through mobile app stores.

In [15], Chen et al. propose mass vetting (MassVet), a lightweight mechanism that is capable of quickly discovering unknown malice at a large scale (e.g., *Google Play Store*). The authors use an app's view structure and navigation relations among views to produce a view graph whose nodes correspond to view items (active widgets) and arcs between nodes show navigation paths. These graphs are further compared with those that original apps on the market have.

The fields of mobile computing (exemplified with mobile apps) and IoT (exemplified with hundreds types or models of device) present many similarities in terms of affordability, wide use, and limited control; thus, they should benefit from each other. Guidelines for vetting mobile apps already exist and would constitute a good source for developing the same for IoT. The next section identifies the gaps in current research related to vetting IoT.

3. Gaps in vetting IoT

Compared to vetting mobile apps (Section 2), there is a major gap in (V)IoT. In addition to Section 1's obstacles that undermine IoT, this gap is also due to the nature of IoT. IoT mixes physical processes with cyber connectivity, making it different from other software-related disciplines [16]. The first set of references that we reviewed are more concerned with the security and privacy of IoT applications than with developing a comprehensive guide for vetting things that would reveal their vulnerabilities. Meanwhile, the second set of references run tests to identify vulnerabilities of IoT devices that are already in operation [16]. There is a consensus that IoT vastly impacts the way we view, use, and interact with smart devices [17]. However, security remains a concern that could turn IoT misuse into a serious

97 problem; such devices collect and use users' personal data without their permissions. McKinsey argues
98 that security may represent the greatest obstacle to IoT growth [18]. In [19], Creager discusses methods
99 for detecting IoT devices' suspicious activities. Devices are monitored for proper behavior, and those
100 that show signs of having been interfered can have their behaviors mitigated and their security issues
101 eliminated.

102 In [20], Celik et al. examine the security and privacy of IoT applications using a program-analysis
103 technique. They use the examples of unlocked doors when nobody is at home and turned-off heaters in
104 cold weather. Both examples illustrate risks for the safety of people and their assets, calling for appro-
105 priate and prompt measures. Despite those risks, Celik et al. do not target IoT but, IoT platforms that
106 are used to develop them and the IoT applications that manage them. The considered IoT platforms are
107 Samsung's SmartThings, Apple's HomeKit, OpenHAB, Amazon AWS IoT, and Android Things.

108 In line with the work of Celik et al., Fernandes et al. [21] look into data protection in the context of
109 IoT, in general, and mobile apps associated with IoT, in particular. They argue that permission-based
110 access control over sensitive data is not enough once an app gains control over data. Apps should make
111 their data-use patterns explicit so that these patterns are enforced (deviations are not allowed) at runtime
112 when sensitive data is used. Fernandes et al.'s system, known as FlowFence, enables robust and efficient
113 flow control between sources and sinks in IoT applications by creating a data-handling mechanism for
114 privacy protection purposes. FlowFence focuses partially on data leakage prevention where the overall
115 IoT device behavior is not considered.

116 In a 2018 report by KEYFACTOR [22], the authors discuss cases of IoT devices that have been subject
117 to attacks although these devices were critical to humans' lives. Vetting IoT devices would have helped
118 prevent or at least reduce such cases by revealing their vulnerabilities ahead of time. In the healthcare
119 domain, in 2017, the US Food and Drug Administration (FDA) recalled 465K pacemakers after discover-
120 ing security flaws that could allow hackers to drain device batteries or send malicious instructions to
121 modify a patient's heartbeat. Vetting pacemakers could have prevented such cases too. Similar news
122 were reported in the automotive industry when a Jeep Cherokee was hijacked turning off the transmis-
123 sion while the vehicle was on the freeway.

124 Among the different works that we reviewed, the works of Palavicini Jr. et al. [23] and Siboni et al. [24]
125 overlap with our objectives. On the one hand, Palavicini Jr. et al. apply symbolic analysis to vet, in a
126 semi-automated way, Industrial IoT (IIoT) firmware using *angr*, a UC Santa Barbara binary analysis
127 framework [25], and *Mecanical Phish*, a component from the same university's cyber reasoning system, to

Table 1: Summary of the common reviewed state-of-the-art research that highlights the scopes, contexts, and main contributions. ● means fully covered, ◐ means partially covered, and □ means uncovered.

Literature	Scope			Context			Contributions
	Security	Privacy	Vetting	Software (Apps)	Hardware	IoT	
He et al. [13]	◐	□	◐	●	□	□	Future directions
Nagappan and Shihab [14]	◐	□	◐	●	□	□	Reviews and recommendations
Chen et al. [15]	●	□	◐	●	□	□	MassVet technique
Celik et al. [20]	●	●	◐	●	□	●	Challenges and opportunities
Fernandes et al. [21]	◐	◐	□	●	□	◐	FlowFence system
Palavicini Jr. et al. [23]	◐	□	●	●	◐	◐	Semi-automatic firmware vetting
Siboni et al. [24]	●	◐	●	●	◐	●	Security testbed framework

128 perform the semi-automated analysis of IIoT. The authors mention that embedded systems and IIoT de-
 129 vices are rapidly increasing in number and complexity. As a result, cyber-physical attacks have become
 130 omnipresent, causing economic and physical damages. They also mention that the firmware for these
 131 systems has become difficult to analyze when searching for malicious functionalities. Their approach
 132 consists of 3 steps: preparation of the firmware image for loading into the *angr* framework, emulation
 133 for verification of discovered vulnerabilities, and analysis of the firmware sample “*angr style*”.

134 On the other hand, Siboni et al. [24] discuss a security testbed for IoT devices that iTrust Lab has
 135 developed. First, they report on the experience of using an IoT search engine, SHODAN [26], to dis-
 136 cover vulnerabilities of IoT devices. It is worth noting that this experience targeted devices that were
 137 already in operation, while we insist that vetting aims at detecting vulnerabilities prior to putting de-
 138 vices into operation. The proposed testbed emulates different types of testing environments that sim-
 139 ulate the activity of multiple sensors and perform predefined and customized security tests along with
 140 advanced security testing analysis. Siboni et al. consider 4 security aspects of IoT devices: *architecture*,
 141 which investigates attacks on hardware and software; *network connectivity*, which investigates attacks on
 142 data distribution; *data collection*, which that investigates data collection with regard to privacy invasion
 143 and information theft; and, finally, *countermeasures and mitigation*, which that investigate how to reduce
 144 the security and privacy risks that IoT devices could cause. According to Siboni et al., the proposed
 145 framework requires further improvements to support the full operational capability so that the entire
 146 IoT environment is considered. In addition, the framework uses different open source tools to simulate
 147 real operational environments and to run security tests. These tools may need a vetting process for it-
 148 self, continuous updates, and special configuration from device-to-device, which makes the proposed
 149 framework complex.

150 Table 1 summarizes the scopes, contexts, and contributions of the reviewed state-of-the-art research.
 151 The table clearly demonstrates that very few papers focus on the IoT vetting process by introducing

152 security testing frameworks. The rest of the research either focuses on mobile apps or introduces a partial
153 vetting process or partially covers security and privacy aspects. In common, the reviewed literature
154 considers testing IoT devices that are already in operation, although this research aims for a vetting
155 framework that detects security vulnerabilities before taking the device into operation.

156 4. Preparing things for vetting

157 According to Crews and Mangal [27], the massive arrival of smartphones and mobile apps has trig-
158 gered a “mini-revolution” in the software engineering discipline. Some concepts, principles, and prac-
159 tices, such as those related to testing, have been adjusted. Touchscreen gestures, location awareness, and
160 orientation need to be tested differently. The same is valid when testing and vetting IoT smart devices.
161 We expect that IoT features in terms of reduced size, restricted connectivity, continuous mobility, limited
162 energy, constrained storage, and additional features that Kamrani et al. [28] discuss will trigger a similar
163 “revolution”. *“The things in the Internet of Things (IoT) can get personal. They can be in your home, your car,
164 and your body. They can make your living and working space smart, and they can be dangerous to your health,
165 safety, and liberty. . . . Is our future a brave new world or a dystopian nightmare? Who decides?”* [29].

166 4.1. Overview

167 By analogy to the NIST definition of the app vetting process [12], the IoT vetting process would be a
168 sequence of activities that an organization would carry out to declare if a thing is “clean” with respect
169 to some IoT safety and security requirements. Our sequence of activities represented in Figure 1 would
170 consist of (i) defining things’ duties that would be subject to vetting, (ii) identifying the vulnerabilities
171 that would affect these duties, (iii) analyzing the impact of these vulnerabilities on these duties, and
172 (iv) developing guidelines and/or recommending techniques to address these vulnerabilities. The first
173 two steps in Figure 1 are already discussed in [9, 10]. Therefore, this article focuses on the impact of
174 vulnerabilities on duties to initiate the development of guidelines against these vulnerabilities.

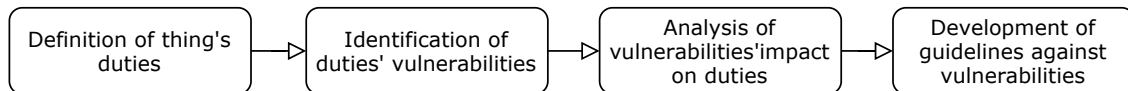


Figure 1: General representation of the VloT process

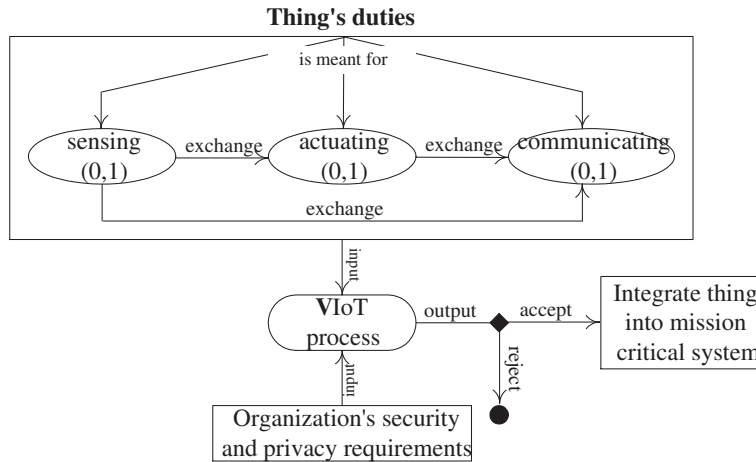


Figure 2: Atomic duties of a thing

175 4.2. Duties of things

176 In a previous work [10, 30], we identified 3 atomic duties that would capture a thing’s capabilities in
 177 terms of *sensing* (collecting and temporarily storing data), *actuating* (processing/acting upon data), and
 178 *communicating* (sharing/distributing data). A duty is either enabled or disabled ((0,1) in Figure 2) accord-
 179 ing to the requirements and needs of the under-development IoT applications. Our duty categorization
 180 overlaps to a certain extent with first, Celik et al.’s sensor-computation-actuator cycle that structures the
 181 design of IoT systems’ apps [20] and second, the IoT device capabilities of [31].

182 Simply put, a thing senses the cyber-physical surrounding so that it generates and (temporarily) stores
 183 (raw) data; a thing actuates data including those that are sensed; and a thing communicates with the
 184 cyber-physical surrounding the sensed and/or actuated data. Accepting data and/or commands from
 185 external parties (e.g., other things) is also taken care of by the communicating duty, but this process is
 186 not further discussed in this report. It is worth noting that a thing’s sensing, actuating, and communi-
 187 cating duties can be composed together as per the following 4 representative cases (other cases, such as
 188 Communicating, Actuating and Communicating, Actuating, Sensing are not discussed):

- 189 1. Sensing, Actuating, Communicating: sensed data are passed on to actuating; and the data that
 190 result from actuation are passed on to communicating for distribution.
- 191 2. Sensing, Actuating: sensed data are passed on to actuating; and the data that result from actuation
 192 are finals.
- 193 3. Sensing, Communicating: sensed data are passed on to communicating for distribution.

194 4. Actuating, Communicating: data that result from actuating are passed on to communicating for
 195 distribution.

196 Figure 3 is the duty model representation referring to one super-class, Duty, which would describe
 197 the common characteristics of all the duties, such as identity (id_duty), name (name_duty), type (type),
 198 and resources to consume when performing the duty (res). Duty is specialized into Sensing, Actuat-
 199 ing, and Communicating. Sensing class describes elements such as frequency (freq_sensing) and sensed
 200 data (data). Actuating class describes elements such as necessary time (latency) and actions (actions)
 201 required to take over data (data). Communicating class describes elements such as time (timestamp), per-
 202 formed action (action), which could be either “receive” or “send”, and the involved things (things) in
 203 this duty. Data class of a thing (thing) describes elements such as the attribute (attribute, e.g., response
 204 time), value (value_data), and validity (validity). Resource class describes elements such as identity (id_res),
 205 name (name_res), and value (value_res). Finally, the Thing class defines the characteristics of the thing that
 206 performs duties such as identity (id_thing) and description (des_thing).

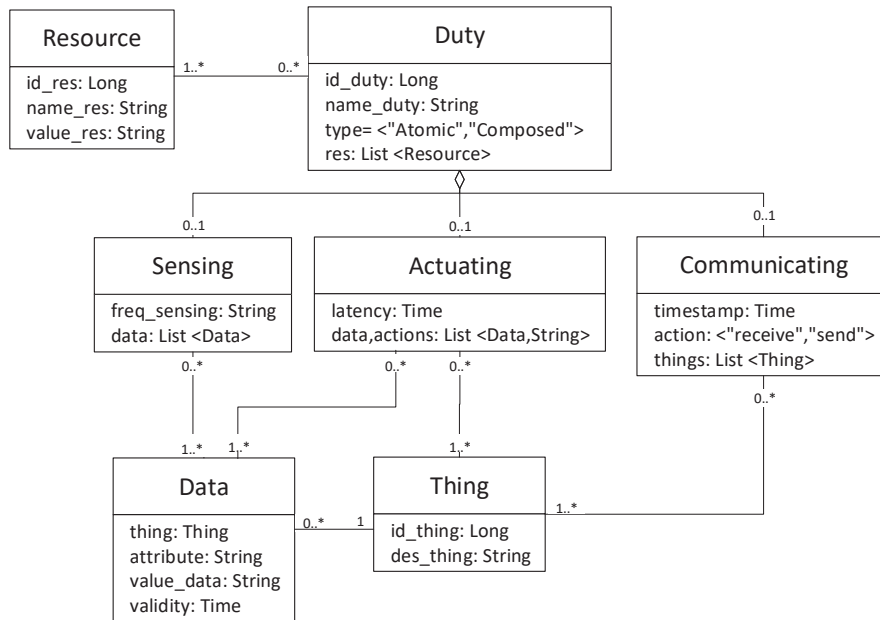


Figure 3: Duty model

207 4.3. Analysis of vulnerabilities of duties

208 As per Figure 1, our vIoT process proceeds with defining the duties of things and then, identifying
 209 potential vulnerabilities that could negatively impact these duties. Although some vulnerabilities that

could be sources of attacks have already been identified in the context of the OWASP IoT project [32] (Table 2), we find these vulnerabilities generic and not related to the duties of things. To address this limited focus, we have developed different questions² that would expose vulnerabilities, as per Table 3, where V_i^x is the i^{th} vulnerability related to either an atomic or a composite duty x , and $x \subset \{s, a, c, sac, sa, sc, ac\}$ (s: sensing, a: actuating, and c: communicating). Only the questions for atomic duties are presented.

Table 2: Common security vulnerabilities in the IoT paradigm in the context of the OWASP IoT project

Vulnerability	Attack surfaces	Summaries
Username enumeration	Administrative interface	Ability to collect a set of valid usernames by interacting with the authentication mechanism
	Device Web interface	
	Cloud interface	
	Mobile applications	
Weak passwords	Administrative interface	Ability to set account passwords to '1234' or '123456', for example Usage of pre-programmed default passwords
	Device Web interface	
	Cloud interface	
	Mobile applications	
Account lockout	Administrative interface	Ability to continue sending authentication attempts after 3 - 5 failed login attempts
	Device Web interface	
	Cloud interface	
	Mobile applications	
Unencrypted services	Device network services	Network services are not properly encrypted to prevent eavesdropping or tampering by attackers
⋮	⋮	⋮

Questions (Q) that can be raised during the vetting of sensing include but are not limited to Q_1 : does sensing target living (e.g., persons) and/or non-living things (e.g., rooms)? What is the sensing about (e.g., ambient temperature, wind speed, and heartbeat)? Q_2 : does sensing target indoor, outdoor, or both? Q_3 : what is the frequency of sensing (e.g., continuously, at regular intervals, or trigger-based)? Q_4 : who configures sensing (e.g., frequencies, service periods, or authorized recipients)? Does configuration need to occur from a specific location and/or using a specific device? Q_5 : what is the resource consumption level of sensing? Also, is there any threshold that would indicate overconsumption and hence, trigger alarms? And, Q_6 : are there traces of tracking sensing using logs, for example? If yes, how are the traces safeguarded?

Questions (Q) that can be raised during the vetting of actuating include but are not limited to Q_1 : can a thing cancel and/or compensate the outcomes of actuating? If yes, does it need any approval? Q_2 : what is the frequency of actuating (e.g., continuously, at regular intervals, or trigger-based)? Q_3 : who configures actuating in terms of frequencies, service periods, etc.? Does configuration have to happen from a specific location and/or using a specific device? Q_4 : what inputs does actuating require? What outputs

²The questions are part of a tip-sheet similar to the one available at <https://tinyurl.com/thpzm3>.

229 does actuating produce? Q₅: are there traces of tracking actuating using logs, for example? If yes, how
230 are these traces safeguarded? And, Q₆: what is the resource consumption level of actuating? And, is
231 there any threshold that would indicate overconsumption and hence, trigger alarms?

232 Questions (Q) that can be raised during the vetting of communicating include but are not limited
233 to Q₁: what interaction protocol does communicating use? Also, is this protocol secured? Q₂: what
234 interaction mode does communicating use (e.g., synchronous *versus* asynchronous and point-to-point
235 *versus* multi-point?) Q₃: what is the frequency of communicating (e.g., continuously, at regular intervals,
236 or trigger-based)? Q₄: who configures communicating in terms of frequencies, mode, service, etc.? Does
237 configuration have to happen from a specific location and/or using a specific device? Q₅: to whom does
238 thing send data (e.g., persons, systems, or both)? Are there guarantees that data recipients do not share
239 it further without approval? Q₆: is data communicated immediately with authorized parties or cached
240 for later sharing? Also, if data is classified, what measures are used for achieving this? Q₇: what is the
241 resource consumption level of communicating? Also, is there any threshold that would indicate over-
242 consumption and hence, trigger alarms? And, Q₈: are there traces of tracking communicating using logs,
243 for example? If yes, how are these traces safeguarded?

244 4.4. Impact of vulnerabilities on duties

245 In this part of the paper, we examine the impact of vulnerabilities on atomic and composite duties.
246 To this end, we capture the successful completion of a duty using a state diagram and then, enrich this
247 diagram with special notations to illustrate how a duty could deviate from this completion because of
248 vulnerabilities. We see deviations as vulnerability's side effects.

249 4.4.1. Atomic duties

250 **Sensing.** Figure 4 is a state diagram that tracks the progress of the sensing duty towards successful com-
251 pletion. Using this diagram's states and transitions, we develop what we refer to as normal-progress
252 cycle of a duty (plain lines in Figure 4). We also enrich this diagram with necessary states and/or
253 transitions to represent deviations from this cycle that correspond to side effects of vulnerabilities
254 (dashed lines in Figure 4).

255 On the one hand, in a normal-progress cycle, the states are not-activated (the sensing duty is not
256 available), activated (the sensing duty is being configured), operated (a composed state where the

Table 3: Sample of vulnerabilities of things per duty

Type	ID	Description
Atomic	V_1^s	Changing sensing frequency(ies) without approval
	V_2^s	Tampering sensing's approved resource consumption-level so, this over consumption gets unnoticed
	V_3^s	Infinite suspension of sensing duty
	V_4^a	Changing actuating frequency without approval
	V_5^a	Altering inputs purposely
	V_6^a	Infinite suspension of actuating duty
	V_7^c	Communicating differently from the claimed frequency
	V_8^c	Using a different protocol from what is claimed
	V_9^c	Unauthorized party proceeds with communicating using a non-acceptable device
	V_{10}^c	Interrupting communicating without resumption
Composite	V_{11}^{SA}	Changing sensing frequency(ies) without approval or (un)synchronized sensing and actuating though it is not mandatory
	V_{12}^{SA}	Interrupting sensing or actuating without resumption
	V_{13}^{SA}	Altering sensed input, which would impact the sensed input to actuate
	V_{14}^{SC}	Changing sensing frequency without approval, which would impact the frequency of the sensed content to communicate
	V_{15}^{SC}	Interrupting sensing or communicating without resumption
	V_{16}^{SC}	Unauthorized party proceeds with sensing and communicating configuration using a non-acceptable device
	V_{18}^{AC}	Changing actuating frequency(ies) without approval or (un)synchronized actuating and communicating though it is not mandatory
	V_{19}^{AC}	Interrupting actuating or communicating without resumption
	V_{20}^{AC}	Altering received input
	V_{21}^{AC}	Unauthorized party proceeds with communicating using a non-acceptable device
	V_{22}^{SCC}	Change in sensing or communicating frequency or resource consumption of the sensing duty
	V_{23}^{SCC}	Infinite suspension of sensing duty or the communicating duty in the sensing duty or the communicating duty in one of the two things
	V_{24}^{SCC}	Altering received input

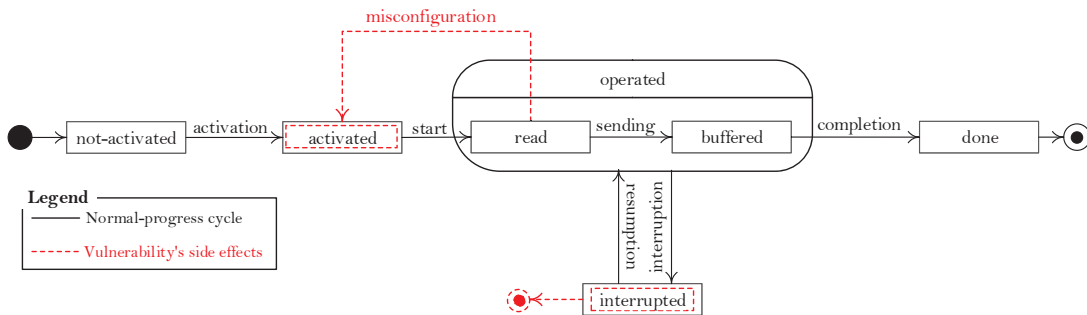


Figure 4: Enriched state diagram of the sensing duty

257 sensing is taking place in terms of reading and buffering³), done (the sensing is complete), and
 258 interrupted (the sensing is put on-hold and then resumed). On the other hand, the transitions con-
 259 necting these states together are activation, start, sending, interruption, resumption, and completion.

260 Vulnerability's side effects correspond to 2 states, namely, activated (the sensing duty is miscon-
 261 figured) and interrupted (the sensing is put on-hold indefinitely), and 1 transition, namely, miscon-
 262 figuration. We associate misconfiguring sensing with 2 vulnerabilities, namely, change in sensing
 263 frequency (V_1^s) and tampering resource-consumption level (V_2^s), and we associate indefinite on-
 264 hold with 1 vulnerability, namely, infinite suspension of sensing (V_3^s).

265 **Actuating.** Similar to sensing, we develop the normal-progress cycle of the actuating duty (plain lines in
 266 Figure 5) and identify the side effects of vulnerabilities that this duty could be subject to (dashed
 267 lines in Figure 5).

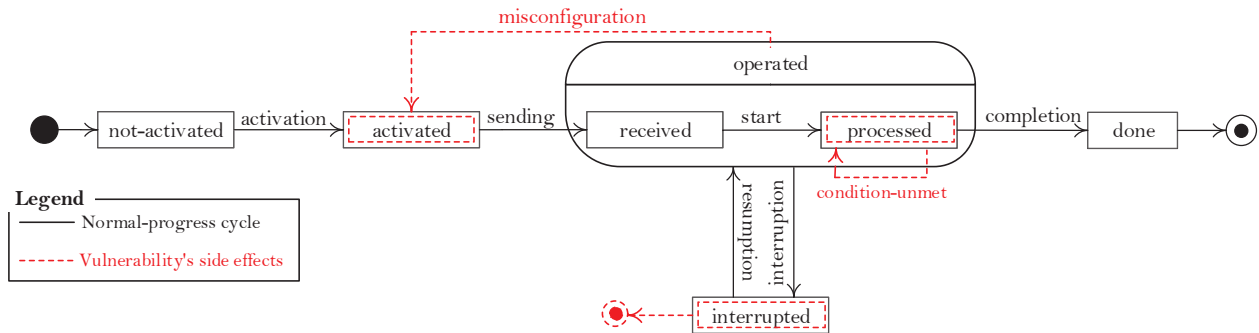


Figure 5: Enriched state diagram of the actuating duty

268 In a normal-progress cycle, on the one hand, the states are not-activated (the actuating duty is not
 269 available), activated (the actuating duty is being configured), operated (a composed state where the
 270 actuating is taking place in terms of receiving and processing data), done (the actuating is com-
 271 plete), and interrupted (the actuating is put on-hold and then resumed). On the other hand, the
 272 transitions connecting these states together are activation, sending, start, interruption, resumption,
 273 and completion.

274 Vulnerability's side effects correspond to 3 states, namely, activated (the actuating duty is being
 275 misconfigured), processed (the actuating operates differently), and interrupted (the actuating is put

³We adjust the sensing behavior reported in [33].

276 on-hold indefinitely), and 2 transitions, namely, misconfiguration and condition-unmet. We asso-
 277 ciate misconfiguring actuating with 1 vulnerability, i.e., change in actuating frequency (V_4^a); we
 278 associate operating differently of the actuating duty with 1 vulnerability, i.e., altering received in-
 279 put (V_5^a); and we associate indefinite on-hold with 1 vulnerability, i.e., the infinite suspension of
 280 actuating (V_6^a).

281 **Communicating.** Like sensing and actuating, Figure 6 shows the normal-progress cycle of the communi-
 282 cating duty (plain lines in Figure 6) and the side effects of vulnerabilities that this duty could be
 283 subject to (dashed lines in Figure 6).

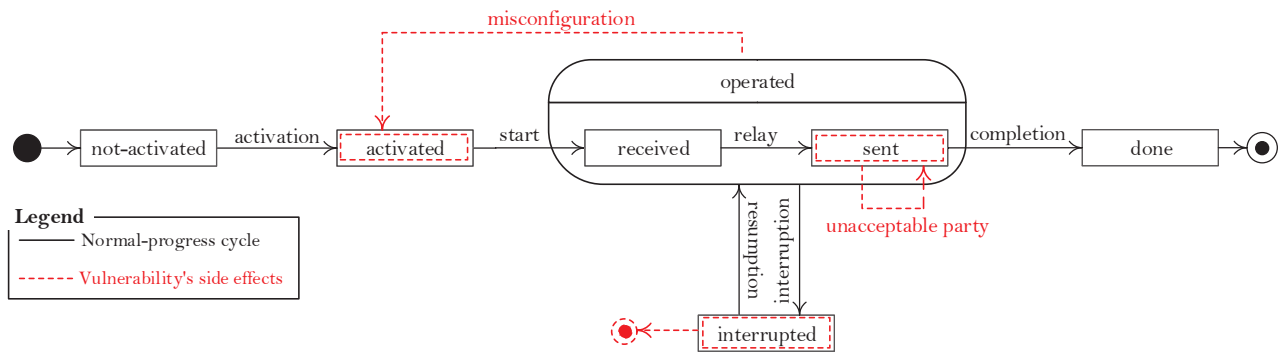


Figure 6: Enriched state diagram of the communicating duty

284 In a normal-progress cycle, on the one hand, the states are not-activated (the communicating duty is
 285 not available), activated (the communicating duty is being configured), operated (a composed state
 286 where the communicating is taking place in terms of receiving and sending data), done (the com-
 287 municating is complete), and interrupted (the communicating is put on-hold and then, resumed).
 288 On the other hand, the transitions connecting these states together are activation, start, relay, inter-
 289 ruption, resumption, and completion.

290 Vulnerability's side effects correspond to 3 states, namely, activated (the communicating duty is
 291 being misconfigured), sent (the community duty shares data with unauthorized party), and inter-
 292 rupted (the communicating is put on-hold indefinitely), and 2 transitions, namely, misconfiguration
 293 and unacceptable-party. We associate misconfiguring communicating with a change in communi-
 294 cating frequency (V_7^c) and a change in communicating protocol (V_8^c); we associate the misuse oper-
 295 ation of the thing's duty with a connected unauthorized party (V_9^c); and we associate the indefinite
 296 on-hold with the infinite suspension of communicating (V_{10}^c).

297 4.4.2. Composite duties

298 **(Sensing,Actuating).** Figure 7 shows the normal-progress cycle of the sensing,actuating composite duty
 299 (plain lines in Figure 7) and the side effects of vulnerabilities that this composite duty could be
 300 subject to (dashed lines in Figure 7).

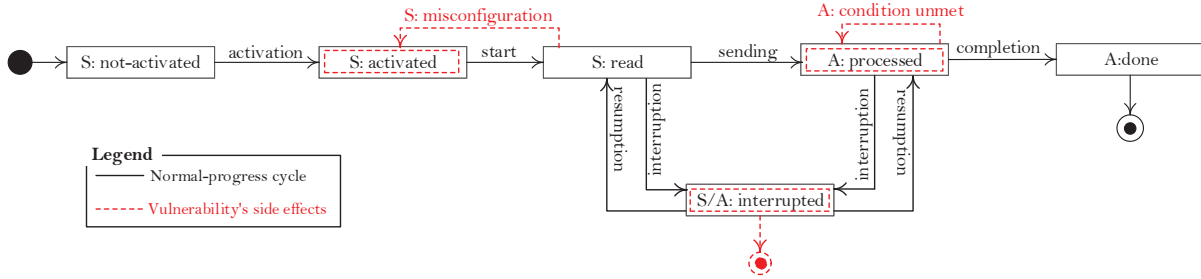


Figure 7: Enriched state diagram of the sensing,actuating composite duty

301 In a normal-progress cycle, on the one hand, the states are S:not-activated (the sensing duty is not
 302 available), S:activated (the sensing duty is being configured), S:read (the sensing duty is taking
 303 place in terms of reading), A:processed (the actuating duty takes over from the sensing duty after
 304 receiving the sensed data for processing), A:done (the actuating duty is complete), and interrupted
 305 (either the sensing duty or the actuating duty is put on-hold and then resumed). On the other hand,
 306 the transitions connecting these states together are activation, start, sending, interruption, resumption,
 307 and completion.

308 Vulnerability's side effects correspond to 3 states, namely, S:activated (the sensing duty is miscon-
 309 figured), S/A: interrupted (either the sensing duty or the actuating duty is put on-hold indefinitely),
 310 and A:processed (the actuating operates differently), and 2 transitions, namely, S:misconfiguration
 311 and A:condition unmet. We associate misconfiguring sensing with a change in sensing frequency (V_{11}^{SA});
 312 we associate indefinite on-hold with 1 vulnerability, namely, the infinite suspension of sensing or
 313 actuating (V_{12}^{SA}); and we associate operating differently of the actuating duty with 1 vulnerability,
 314 namely, altering sensed input (V_{13}^{SA}).

315 **(Sensing,Communicating).** Figure 8 shows the normal-progress cycle of the sensing,communicating com-
 316 posite duty (plain lines in Figure 8) and the side effects of vulnerabilities that this composite duty
 317 could be subject to (dashed lines in Figure 8).

318 In a normal-progress cycle, on the one hand, the states are S:not-activated (the sensing duty is not

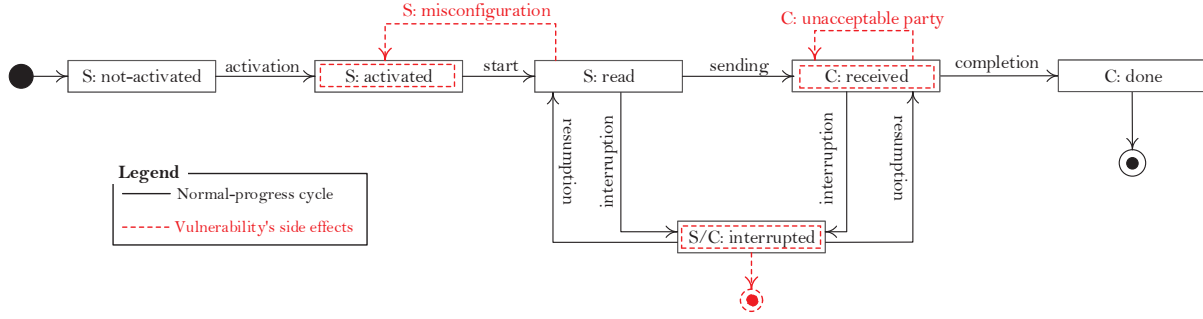


Figure 8: Enriched state diagram of the sensing, communicating composite duty

available), S:activated (the sensing duty is being configured), S:read (the sensing duty is taking place in terms of reading), C:received (the communicating duty is taking place in terms of receiving data), and S/C: interrupted (either the sensing duty or the communicating duty is put on-hold and then, resumed), and C:done (the communicating duty is complete). On the other hand, the transitions connecting these states together are activation, start, sending, interruption, resumption, and completion. Vulnerability's side effects correspond to 3 states, namely, S:activated (the sensing duty is misconfigured), C:received (the communicating duty shares data with unauthorized party) and S/C:interrupted (either the sensing duty or the communicating duty is put on-hold indefinitely), and 2 transitions, namely, S:misconfiguration and C:unacceptable party. We associate misconfiguring sensing with the change in sensing frequency by an unauthorized person (V_{14}^{SC}); we associate indefinite on-hold with 1 vulnerability, i.e., infinite suspension of sensing or communicating (V_{15}^{SC}); and we associate the misuse operation of the communicating duty with a connected unauthorized party (V_{16}^{SC}).

(Sensing, Communicating, Communicating). Figure 9 shows the normal-progress cycle of the sensing, communicating, communicating composite duty (plain lines in Figure 9) and the side effects of vulnerabilities that this composite duty could be subject to (dashed lines in Figure 9).

In a normal-progress cycle, on the one hand, the states are S:not-activated (the sensing duty is not available), S:activated (the sensing duty is being configured), S:read (the sensing duty is taking place in terms of reading data), C1:received (the actuating duty is taking place in the first thing in terms of receiving data), C1/C2:synchronized (the communicating duty is synchronized between the first and the second things), C2:received (the actuating duty is taking place in the second thing in terms of receiving data), C2:done (the communicating duty is complete) and S/C1/C2:interrupted

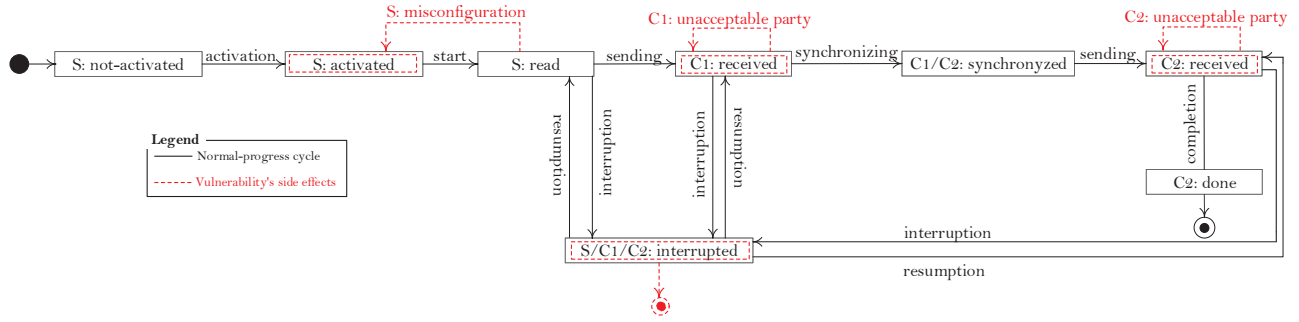


Figure 9: Enriched state diagram of the sensing,communicating,communicating composite duty

(either the sensing duty or the communicating duty in one of the two things is put on-hold and then resumed). On the other hand, the transitions connecting these states together are activation, sending, start, interruption, resumption, relay, and synchronizing.

Vulnerability's side effects correspond to 4 states, namely, S:activated (the sensing duty is misconfigured), C1: received (the communicating duty in the first thing shares data with unauthorized party), C2: received (the communicating duty in the second thing shares data with unauthorized party) and S/C1/C2: interrupted (either the sensing duty or the communicating duty in one of the two things is put on-hold indefinitely), and 3 transitions, namely, S:misconfiguration, C1: unacceptable party and C2: unacceptable party. We associate misconfiguring sensing with the change in sensing or communicating frequency, or resource consumption (V_{22}^{SCC}); we associate an indefinite on-hold with 1 vulnerability, i.e., the infinite suspension of sensing duty or the communicating duty within the sensing duty or the communicating duty in one of the two things (V_{23}^{SCC}), and operating differently of the communicating duty in one of the two things with altering received input (V_{24}^{SCC}).

5. Guiding framework for vetting things

This section presents our proposed \mathcal{V} IoT framework (Section. 5.1) and, then, the system that was implemented accordingly (Section. 5.2).

5.1. Architecture

Figure 10 identifies the modules that constitute our framework for vetting things. Vetting ensures that a thing's behavior is in line with its expected behavior (Section 4.4) when tested against vulnerabili-

ties that its duties can be subject to (Section 4.3). The progress of completing the vetting in the framework goes over 5 stopovers.

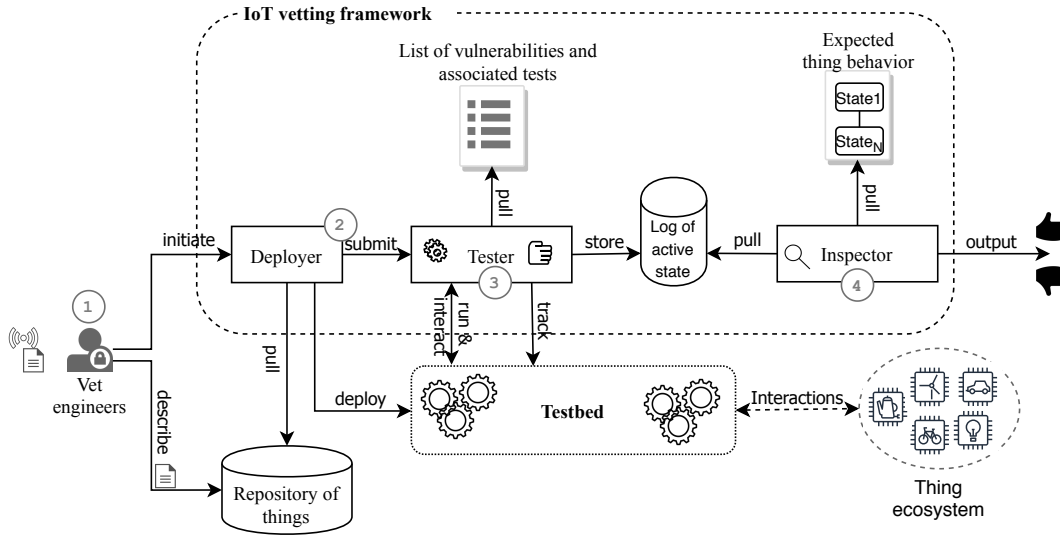


Figure 10: Modules of the \mathcal{V} IoT framework

safe

In stopover (1), the vet engineer enriches the description of the thing to vet with details about its atomic/composite duties and then, publishes this description on the repository of things. In stopover (2), the vet engineer deploys the thing on the testbed allowing to run different tests and to support interactions with peers and the Tester module. The deployment is taken care by the deployer module that also extracts the thing’s descriptions of duties from the repository of things and submits them to the tester module, thereby initiating the third stopover (3). The tester module (i) retrieves the list of known vulnerabilities for the thing based on its duties, (ii) selects the necessary tests⁴ associated with these vulnerabilities and finally, (iii) runs the selected tests. During the tests, the testbed is constantly monitored thanks to the tester module that tracks and stores both the states of the thing and the events in a dedicated log. Upon completing the tests that correspond to stopover (4), the following occurs; the inspector module pulls the behavior of the under-vetting thing from the log and its expected behavior as defined in Section 4.4. Then, it compares the behavior built upon the tests to the expected one: if no vulnerabilities’ side-effects are detected then the thing is declared **safe**; otherwise, it is declared **unsafe**.

⁴Tests could be automatic, semi-automatic, or manual.

376 5.2. Implementation and experiments

377 A system demonstrating the \mathcal{V} IoT framework was developed using Python 3, Java 14, JavaScript,
 378 MQTTLens [34] and Node-RED [35]. MQTTLens is an add-on for the Chrome browser allowing to pub-
 379 lish messages to an MQTT broker, to subscribe to MQTT topics and to receive messages. This latter
 380 is a flow-based programming paradigm in which a variety of processes called “nodes” are connected
 381 together to form applications called “flows”, instead of writing code. Node-RED runs on local worksta-
 382 tions, the cloud, and edge devices. It has become an ideal tool for the Raspberry Pi and other low-cost
 383 hardware. Node-RED runtime is built on top of Node.js and takes its full advantage in terms of event-
 384 driven, non-blocking I/O model. It can be used to build and deploy applications due to its browser-
 385 based editor by wiring hardware devices, APIs and online services. Node-RED implements the testbed
 386 and includes an MQTT server so that IoT devices could be easily connected via port 1883 using Transport
 387 Layer Security (TLS). We considered sensors as things to vet. The vet engineer interacts with the sensors
 388 running on the testbed using JSON requests via the deployer module (i.e., MQTTLensTool). The way a
 389 sensor was implemented consists of 3 connected objects: living.lights, bed.alarm, and kitchen.aircondition.
 390 They were created and deployed by the Node-RED server. On top of Node-RED, other parts (i.e., the
 391 tester and the inspector) were implemented using Python language (Spyder Tool V3.7 [36]).

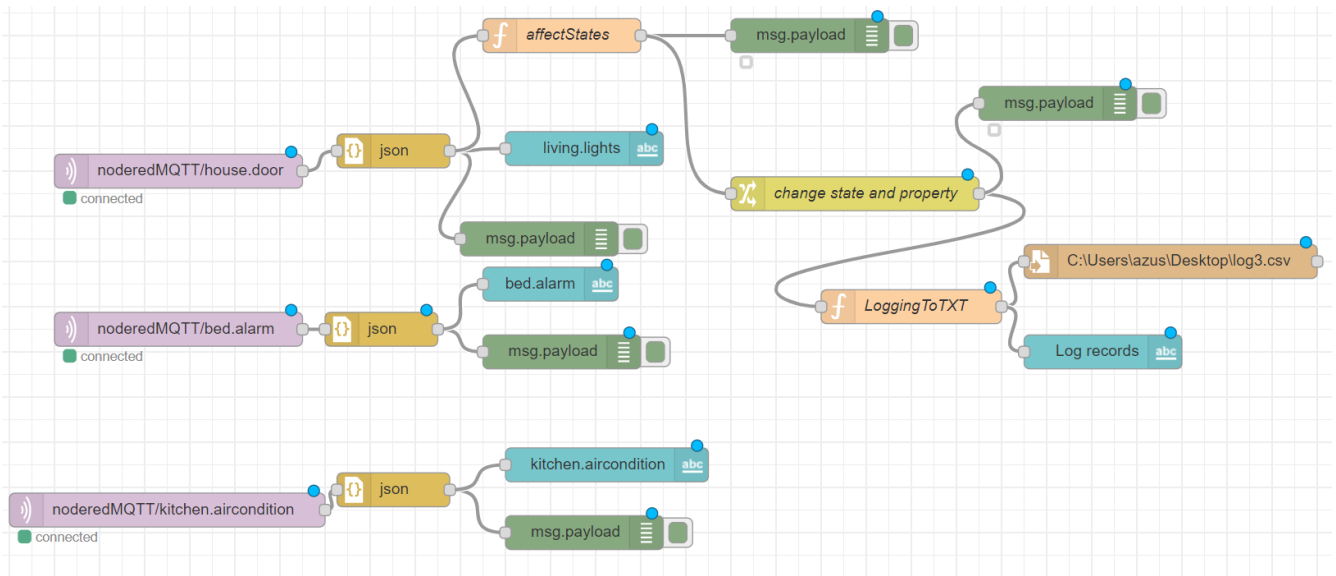


Figure 11: Things deployed in the Node-RED testbed of the \mathcal{V} IoT framework

392 According to Figure 11 (the testbed), we used six node types (i) *mqtt input* node allows users to con-
 393 figure an MQTT service and the topic to publish about. As a result, a node could receive messages with
 394 the exact same JSON string that was sent via MQTTLens (as shown in Figure 12). (ii) *JSON* node which

395 parses messages that MQTT input node generates when it receives an MQTT message and converts it to
 396 a JavaScript object. (iii) *function* node receives a message object as input and can return 0 or many mes-
 397 sages objects as output. A message object must have at least a payload property (`msg.payload`). (iv) *debug*
 398 node displays the received messages within the flow in the panel. (v) *text* node shows messages within
 399 the flow on a slider and as a text string. Finally, (vi) *change* node changes a message payload or adds
 400 new properties. All calls to the testbed’s flow are tracked and stored by the Tester in the log of the active
 401 states database consisting on a CSV file. Next, the Inspector module compares the stored active states to
 402 the expected ones to check the possible existence of any vulnerability.

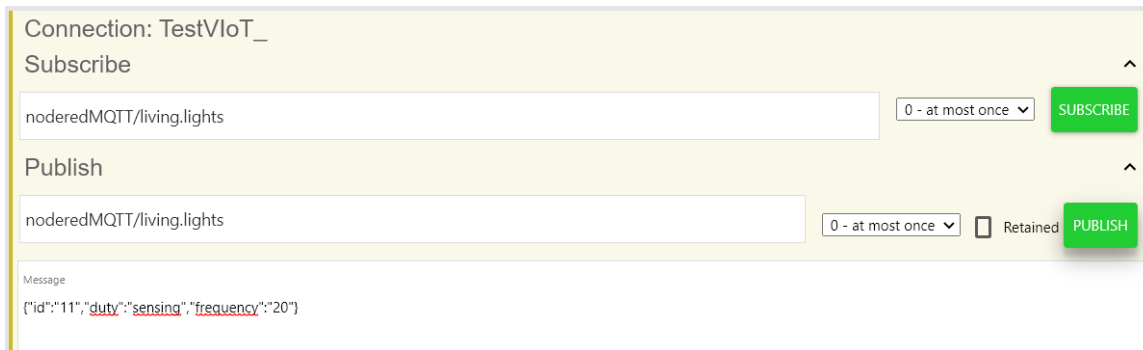


Figure 12: JSON request submission via MQTTLens

403 To illustrate a vulnerability, we considered sensing duty and set its frequency to every time inter-
 404 val. A vulnerability would consist of communicating differently from the claimed frequency (V_1^s) by
 405 using the “change” node. Thus, during the experiment, the Python program related to sensing duty
 406 was launched in order to check whether the stored states were as expected. If not, it displayed a notifi-
 407 cation of a vulnerability. In the following, we considered 2 series of vulnerability-related experiments:
 408 vetting duration and vetting efficiency. These experiments run on Windows 10 Asus TUF Gaming i7-
 409 8750H @ 2.206Hz, 12Go. Before all, the VloT engineer sends a JSON request to a living.lights’s sensor via
 410 MQTTLens Tool as per Figure 12.

- 411 1. Vetting duration: to assess the framework’s time performance for detecting vulnerabilities. We
 412 computed the time of sending a request and the time of detecting a vulnerability. We considered
 413 10 requests with an 2-second time interval between each one, where each request was tampered
 414 by changing frequency vulnerability. Then, we recorded the time that the inspector took to process
 415 the vetting so, that vulnerabilities were detected. As per Figure 13, we noticed that launching vio-

416 lations does not affect much the vetting latency since the active states are checked by the Inspector
 417 module.

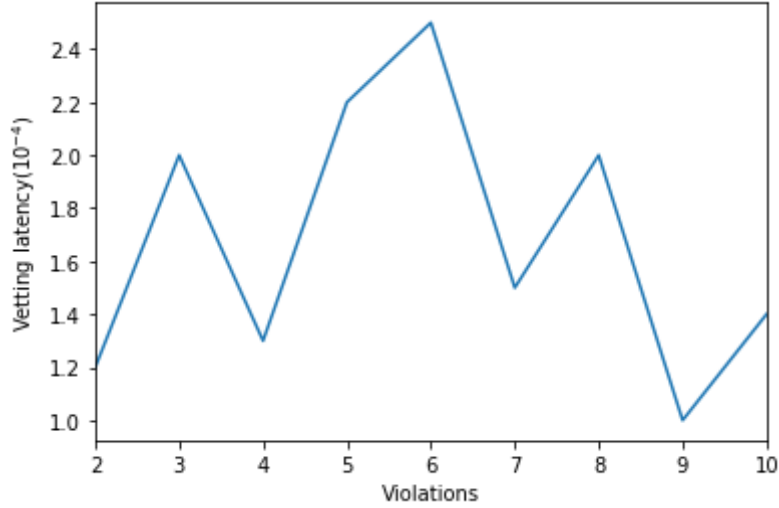


Figure 13: Latency-based VloT framework performance

418 2. Vetting efficiency: to assess the framework’s capacity to detect vulnerabilities based on precision
 419 and recall. The precision expresses the ability of the Inspector module to detect correct (i.e., that
 420 actually happened) vulnerabilities only. It is defined by calculating the ratio of the number of true
 421 vulnerabilities among the launched vulnerabilities as per Equation (1), where TV is the number of
 422 true vulnerabilities, and FV is the number of detected false vulnerabilities.

$$Precision = \frac{TV}{TV + FV} \quad (1)$$

423 The recall is defined as the ratio of true vulnerabilities over the total number of detected vulnera-
 424 bilities by the Inspector, as per Equation (2), where FN is the number of missing true vulnerabilities
 425 that are not detected. It represents the probability of detecting all true vulnerabilities among the
 426 detected vulnerabilities.

$$Recall = \frac{TV}{TV + FN} \quad (2)$$

427 In these experiments, we relied on the vulnerabilities used in the first series (Figure 13). Then, we
 428 observed the behavior of the Inspector module in terms of number of detected true/false vulnera-
 429 bilities, and the number of non-detected ones to determine the precision and recall values (Table 4).

430 As per Table 4, we noted that the VloT framework is efficient by achieving a precision close to 1 in

Table 4: Precision/recall results

Vulnerabilities	1	2	3	4	5	6	7	8	9	10	Total
TV	1	1	0	1	1	1	1	1	1	1	9
FV	0	1	0	1	2	0	0	0	0	0	4
FN	0	0	1	0	0	0	0	0	0	0	1
Precision	1	0.5	0	0.5	0.33	1	1	1	1	1	0.9
Recall	1	1	0	1	1	1	1	1	1	1	0.9

431 most of vulnerabilities (same for recall values). We also noticed that there is only one missing vulnerabil-
 432 ity among 10 (the third vulnerability). The efficiency of the VIoT framework in terms of precision/recall
 433 is explained by the presence of the inspector that browses instantly the active states of the connected
 434 things, where each is checked separately whether it is as expected or not.

435 6. Conclusion

436 This paper discussed the importance of vetting things prior to their integration into systems, some
 437 of which are critical. The objective is to verify claims of providers about the safety of things as some
 438 could be subject to vulnerabilities that could be taken advantage of for non-beneficial practices. The
 439 IoT paradigm imposes a new way of how determining security, confidentiality, identity, and privacy
 440 should be approached. By analogy to mobile apps vetting, we identified the vulnerabilities that could
 441 impede things' duties from successful completion and then illustrated how these vulnerabilities could
 442 be detected. The duties are referred to as sensing, actuating, and communicating; they have been exam-
 443 ined separately in the context of atomic duties and then combined, in the context of composite duties.
 444 The outcome of vetting either ensures thing's trustworthiness when their behaviors meet the expected
 445 execution cycles or declares them as unsafe. Future work will target completing the testbed by vetting
 446 more things and more duties, whether atomic and composite, and examining the concept of provable
 447 vetting so that things would be accountable for the duties they perform.

448 References

- 449 [1] Gunduz, M.Z., Das, R.: Cyber-security on smart grid: Threats and potential solutions. *Computer Networks*
 450 169, 107094 (2020), <https://doi.org/10.1016/j.comnet.2019.107094>
- 451 [2] Statista: Mobile app usage—Statistics & Facts (2019), [https://www.statista.com/topics/1002/
 452 mobile-app-usage/](https://www.statista.com/topics/1002/mobile-app-usage/), [Online; accessed 20-March-2020]
- 453 [3] Ashford, W.: Less than half of firms able to detect IoT breaches (2019), [https://www.computerweekly.
 454 com/news/252455834/Less-than-half-of-firms-able-to-detect-IoT-breaches-study-shows](https://www.computerweekly.com/news/252455834/Less-than-half-of-firms-able-to-detect-IoT-breaches-study-shows), [Online;
 455 accessed 16-June-2020]

- 456 [4] Middleton, P., Tully, J., Kjeldsen, P.: Forecast: The internet of things, worldwide, 2013 (2013), <http://www.gartner.com/doc/2625419/>, Report of Gartner. [Online; accessed 19-August-2020]
- 457
- 458 [5] Ko, E., Kim, T., Kim, H.: Management platform of threats information in iot environment. *Journal of Ambient Intelligence and Humanized Computing* 9(4), 1167–1176 (2018)
- 459
- 460 [6] DZone: The Internet of Things, Application, Protocols, and Best Practices. Tech. rep. (<https://dzone.com/guides/iot-applications-protocols-and-best-practices>, 2017 (visited in May 2017))
- 461
- 462 [7] Elrawy, M.F., Awad, A.I., Hamed, H.F.A.: Intrusion detection systems for IoT-based smart environments: a survey. *Journal of Cloud Computing* 7(1), 21 (2018), <https://doi.org/10.1186/s13677-018-0123-6>
- 463
- 464 [8] Quirolgico, S., Voas, J., Kuhn, R.: Vetting Mobile Apps. *IT Professional* 13(4) (2011)
- 465
- 466 [9] Maamar, Z., Kajan, E., Asim, M., Baker Shamsa, T.: Open challenges in vetting the internet-of-things. *Internet Technology Letters* 2(5), e129 (2019), <https://doi.org/10.1002/itl2.129>
- 467
- 468 [10] Qamar, A., Asim, M., Maamar, Z., Saeed, S., Baker, T.: A quality-of-things model for assessing the internet-of-things' nonfunctional properties. *Transactions on Emerging Telecommunications Technologies* p. e3668, <https://doi.org/10.1002/ett.3668>
- 469
- 470 [11] Papazoglou, M., Pohl, K., Parkin, M., Metzger, A. (eds.): *Service Research Challenges and Solutions for the Future Internet - S-Cube - Towards Engineering, Managing and Adapting Service-Based Systems*, Lecture Notes in Computer Science, Vol. 6500. Springer (2010)
- 471
- 472
- 473 [12] Ogata, M., Franklin, J., Voas, J., Sritapan, V., Quirolgico, S.: Vetting the Security of Mobile Applications (April 2019), <https://doi.org/10.6028/NIST.SP.800-163r1>, DRAFT NIST Special Publication 800-163 (Revision 1). [Online accessed 20-March-2020]
- 474
- 475
- 476 [13] He, D., Chan, S., Guizani, M.: *Mobile Application Security: Malware Threats and Defenses*. *IEEE Wireless Communications* 22(1) (February 2015)
- 477
- 478 [14] Nagappan, M., Shihab, E.: Future Trends in Software Engineering Research for Mobile Apps. In: *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'2016)*. Vol. 5. Suita, Osaka, Japan (March 2016)
- 479
- 480
- 481 [15] Chen, K., Wang, P., Lee, Y., Wang, X., Zhang, N., Huang, H., Zou, W., Liu, P.: Finding Unknown Malice in 10 Seconds: Mass Vetting for New Threats at the Google-play Scale. In: *Proceedings of the 24th USENIX Conference on Security Symposium (SEC'2015)*. Washington, D.C., USA (2015)
- 482
- 483
- 484 [16] Ali, B., Awad, A.I.: Cyber and physical security vulnerability assessment for IoT-Based smart homes. *Sensors* 18(3), 817 (2018), <https://doi.org/10.3390/s18030817>
- 485
- 486 [17] Hassan, A.M., Awad, A.I.: Urban transition in the era of the Internet of Things: Social implications and privacy challenges. *IEEE Access* 6, 36428–36440 (2018), <https://doi.org/10.1109/ACCESS.2018.2838339>
- 487
- 488 [18] Bauer, H., Burkacky, O., Knochenhauer, C.: *Security in the Internet of Things* (2020), <http://www.mckinsey.com/>, [Online; accessed 08-March-2020]
- 489
- 490 [19] Creager, L.: How can Anomalous IoT Device Activity be Detected? tinyurl.com/y9v5lgfq (Downloaded in

- 491 November 2018)
- 492 [20] Celik, Z.B., Fernandes, E., Pauley, E., Tan, G., McDaniel, P.: Program analysis of commodity iot applications
493 for security and privacy: Challenges and opportunities. *ACM Comput. Surv.* 52(4) (Aug 2019), [https://doi.
494 org/10.1145/3333501](https://doi.org/10.1145/3333501)
- 495 [21] Fernandes, E., Paupore, J., Rahmati, A., Simionato, D., Conti, M., Prakash, A.: FlowFence: Practical Data
496 Protection for Emerging IoT Application Frameworks. In: *Proceedings of the 25th USENIX Security Sympo-
497 sium (USENIX Security'2016)*. Austin, TX, USA (2016)
- 498 [22] KEYFACTOR: The Impact of Unsecured Digital Identities (2020), <https://www.keyfactor.com/>, [Online;
499 accessed 08-March-2020]
- 500 [23] Palavicini Jr., G., Bryan, J., Sheets, E., Kline, M., San Miguel, J.: Towards Firmware Analysis of Industrial
501 Internet of Things (IIoT) - Applying Symbolic Analysis to IIoT Firmware Vetting. In: *Proceedings of the 2nd
502 International Conference on Internet of Things, Big Data and Security (IoTBDS'2017)*. pp. 470–477. Porto,
503 Portugal (2017), <https://doi.org/10.5220/0006393704700477>
- 504 [24] Siboni, S., Sachidananda, V., Meidan, Y., Bohadana, M., Mathov, Y., Bhairav, S., Shabtai, A., Elovici, Y.:
505 Security testbed for internet-of-things devices. *IEEE Transactions on Reliability* 68(1), 23–44 (March 2019),
506 <https://doi.org/10.1109/TR.2018.2864536>
- 507 [25] Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., Feng, S., Hauser, C.,
508 Kruegel, C., Vigna, G.: SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In: *Pro-
509 ceedinmgs of the Symposium on Security and Privacy (SP'2016)*. pp. 138–157. San Jose, CA, USA (2016),
510 <https://doi.org/10.1109/SP.2016.17>
- 511 [26] The search engine for the Internet-connected devices: <https://www.shodan.io>, [Online; accessed 19-August-
512 2020]
- 513 [27] Crews, B., Mangal, S.: IoT and it's Impact on Testing, [https://www.getzephyr.com/resources/
514 whitepapers/iot-and-its-impact-testing](https://www.getzephyr.com/resources/whitepapers/iot-and-its-impact-testing), [Online; accessed 19-July-2020]
- 515 [28] Kamrani, F., Wedling, M., Rodhe, I.: Internet of Things: Security and Privacy Issues. Tech. rep., FOI Swedish
516 Defence Research Agency, Defence and Security, Systems and Technology (2019)
- 517 [29] Orman, H.: You Let That In? *IEEE Internet Computing* 21(3) (2017)
- 518 [30] Maamar, Z., Baker, T., Sellami, M., Asim, M., Ugljanin, E., Faci, N.: Cloud vs edge: Who serves the Internet-
519 of-Things better? *Internet Technology Letters* 1(5), e66 (2018), <https://doi.org/10.1002/itl12.66>
- 520 [31] NIST, NSA: CISCO IoT Industrial Ethernet and Connected Grid Switches running IOS (Downloaded in
521 November 2018), report Number: CCEVS-VR-10692-2015, March 11 2016
- 522 [32] OWASP: OWASP Internet of Things (2020), <https://owasp.org/www-project-internet-of-things/>, [On-
523 line; accessed 08-March-2020]
- 524 [33] Costa, B., Pires, P., Delicato, F., Li, W., Zomaya, A.: Design and Analysis of IoT Applications: A Model-Driven
525 Approach. In: *Proceedings of the 2016 IEEE 14th International Conference on Dependable, Autonomic and*

- 526 Secure Computing, the 14th International Conference on Pervasive Intelligence and Computing, and the
527 2nd International Conference on Big Data Intelligence and Computing and Cyber-Science and Technology
528 Congress (DASC/PiCom/DataCom/CyberSciTech'2016). Auckland, New Zealand (2016)
- 529 [34] MQTTLens: MQTTLens (2020), [https://chrome.google.com/webstore/detail/mqttlens/
530 hemojaaeigabkcookmlgmdigohjobjm](https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkcookmlgmdigohjobjm), [Online; accessed 19-July-2020]
- 531 [35] Red, N.: Node Red (2020), <https://nodered.org/>, [Online; accessed 19-July-2020]
- 532 [36] Spyder: Spyder Website (2020), <https://www.spyder-ide.org/>, [Online; accessed 22-July-2020]