Analysis of IoT devices via API Exploitation and Model Extraction

Alexandre Eraclides
Alexandre.gpe@hotmail.com
Instituto Superior Técnico, Lisboa, Portugal
November 2018

Abstract

In the last few years there has been a great expansion in the development of the Internet of Things (IoT) on different areas, with a great increase in the production of new devices like smartwatches, smart TVs, Internet refrigerators, smart light bulbs and many more. The great majority of the IoT consumers are not aware that many of these products are rushed to the market and extensive testing is not always performed. This problem will get worse in the future, due to the foreseen increase of IoT devices and their complexity, which will also increase the possibility of errors in the software of these devices.

Over the course of this thesis we developed a new tool that automatically tests the IoT devices through a combination of method sequences and black-box testing. Also at the end of the testing, the tool generates a model that exhibits the sequences with problems. In order for the user to have a better understanding of these problems, the tool produces logs that together with the model allow him to recreate the problems found during the tests.

The tool tests every method in different internal states of the IoT device that it is able to reach. These sequences have the objective of setting the device into different internal states to exercise the methods in different environments. The goal of this test methodology is to attempt to maximize the number of scenarios that correspond to unintended behavior of the devices.

**Keywords:** IoT, API testing, black-box testing, model inference, fuzzing

1. Introduction

The potential of the Internet of Things (IoT) is becoming increasingly more important in the world market. Initially it gained greater visibility in the domestic segment associated to electrical appliances and to personal use technology, but new emerging projects such as self-driving cars, health care monitoring, smart cities or home automation are generating new deployment opportunities.

In a near future, the rate that the IoTs will come up and their diversity will be so extraordinary that they will be present in almost every aspect of our lives, sharing and collecting data. The effect of the IoTs will go beyond the lives of each person and will have a great impact in every branch of our society, like the industry, health care, economy, cities and many others.

In the coming years, the IoT devices will start to be used in systems where the consequences of a hacking attack can be catastrophic. Since lives and infrastructures will be at risk, if these systems are hacked they can cause accidents on driverless cars, pacemakers malfunction, traffic jams in smart cities or fires at homes. These are some scenarios that are not difficult to imagine. Security breaches are probably the biggest concern and there is no doubt that risk levels will increase with the growth of IoT deployments.

Usually the IoT APIs either use REST or SOAP protocols to communicate over the internet. Although there is not much work done in the field of testing APIs of IoT devices through their respective protocols, testing has been largely studied for general software. One way that an API can be tested, is to individually test each method call. To this end black-box testing is an effective technique, that does not require access to the source code and has the goal of testing the program or method through its input. There are some tools like Pulsar[1], AFL[3], libfuzz1, Sulley2/Boofuzz3, Autofuzz2, Syntribos4, PyJFuzz and many others, that were developed to test specific objectives like an internet protocol, local program coverage or

---

1 https://llvm.org/docs/LibFuzzer.html access on 12/14/2017
2 https://github.com/OpenRCE/sulley on 12/14/2017
3 https://github.com/jtpereyda/boofuzz on 12/14/2017
4 https://github.com/openstack/syntribos access on 12/14/2017
5 https://github.com/mseclab/PyJFuzz access on 12/14/2017
even REST APIs.

In order to test non-intended behaviours of IoT devices, we develop a tool that tests the devices through the API REST communication protocol, thus excluding any physical tests. This tool tries to solve some of the existing problems in the current tools. It is open source, it has minimal interactions with the user during the tests to the IoT device, it automatically generates tests and execute them to the IoT and at the end it produces a model that becomes the “roadmap” to recreate the problems found. In order for this tool to work it requires three files from the user, one file to describe the API of the IoT, another file to describe how to put the IoT device in the right state (the “reset”) and the last one to configure the tool to the user preferences.

The testing to the API device will be done in two major steps. The first step is the “setup”, on which the device is set in a certain internal state and some values are retrieved from the IoT device. Followed by the second step which it runs the automatic tests, using all the existing method calls in the device API. This way, it is possible to check the correct behavior of the methods. In addition, it is also possible to evaluate how the device behaves calling the methods in different internal states.

2. The Solution

In order to test IoT devices through their API’s we decided to develop a new tool. For this tool we wanted something different from what the existing solutions do, but at the same time something that was capable of producing good and new results, from the perspective of discovering problems in the software behind the IoT devices. Some of the established objectives for this tool were: it had to be easy to use, the testing process should be automated, it would be open source and at the end it would produce a model. This model has the objective of helping the user to understand and recreate a certain error.

The testing process of this new testing tool will be based on two methodologies. The first methodology is fuzzing. Fuzzing will be applied to the payloads and url of the API methods in the IoT device, to guarantee that nothing that is outside of the parameters stated in the API specification is accepted by the IoT device. This situations may range from inputs that are outside of the parameters described, special characters to the software in general, numbers where should be strings and vice-versa, etc. To the fuzzing process in our tool we use three different tools, two are open source while the last was developed by us. The first tool is Radamsa, that is a specialized fuzzing program that receives a seed (it can be a normal/valid value) and returns the fuzzed seed. This program is useful to do some general fuzzing on the API method parameters, this allow us to verify if something outside of the valid values is accepted. The second tool is Pyj fuzz, that is specialized in fuzzing payloads in the JSON format. This is useful to us, because not only it allow us to fuzz the parameters in the payload but also it fuzzes the structure of the JSON, it allows us to verify if a sent payload that has an invalid structure for a JSON, is rejected or accepted by the IoT device. The reason why we used two tools for the fuzzing (the Radamsa and the Pyj fuzz) is that the Pyj fuzz is ideal to target the structure of a JSON, but it doesn’t always produce a good fuzzing on the parameters of the payload. For that reason we introduced the tool Radamsa to do specific fuzzing to the parameters of the payload. The last program was developed by us and it has the objective of testing the limits of the parameters in the payload, based on information retrieved by the specification.

The second methodology consists of using sequences of method calls to put the IoT device in different internal states. What we pretend with this is to check if certain internal states can have an impact in accepting or rejecting API calls that have invalid parameters. These internal states can influence the API calls in two ways. The first is that a specific parameter in the internal state of the IoT device, may be required to have a certain value in order for the API call to be accepted. The second is a quantitative internal state, which means that the internal state has a internal variable that is dynamic, for example a counter, only when this variable meets some kind of criteria is the API call accepted (for example counter needs to be below a certain value or above a certain value). If these dependencies are not properly checked and the API calls are made regardless of the fact of the device being in the correct state or not, then a behavior which was not foreseen may happen.

Before we can run the tool, it is necessary to prepare and configure three JSON files. One for detailing The API of the IoT device, other to inform our tool of how to put the IoT device in the initial state and the last is to configure the tool to the user intentions. This files will be crucial for the tool to know how to interact with the IoT device using the correspondent API, to set up the initial state before every test and also to allow the user to set up some parameters that may that may help the tool in the fuzzing process, like the number of request that the IoT device accepts per minute.

The general architecture of this tool is composed of seven modules. The first module is the Json reader, that is responsible for parsing the JSON files, in order to extract some vital information for the operation of the tool, for example the API
methods (name of the methods, input parameters, ranges of the inputs, type of the inputs, etc) to be called in the testing phase. The second module is the Fuzzer, that is responsible for all the testing process to the IoT. Using tools and functions to generate the fuzzing of the payload and the url to test in the IoT device. The third module is the Caller, that is responsible for all the communications between the program and the IoT device and also for resetting the IoT device to the initial state. The fourth module is the Logger, that is responsible for logging the response of the IoT device. Also it will filter the responses that are interesting and discard the ones that are not interesting for the testing context. The fifth module is the Sequence Producer, as the name implies this module is responsible for building the sequence that will be used by the tool to test the IoT device. The sixth module is the Model Producer, that will use some specific logs to produce the model. The last module is the main module that runs the programs and connects all of the previous models. In the figure 1 we can see the architecture of the system.

3. Results
Our tool was tested with multiple IoT devices available in the market and in this section we discuss the results obtained for a specific device. The issues found are non-conformance issues with respect to the specification (and not security vulnerabilities) and were communicated to the developers and fixed in due time. The assignment that takes longer is the generation of the three configuration files, which is the only task that requires human intervention in our tool. The tool generates a high amount as well as diversified tests and the time it takes to test the devices is proportional to the depth set to the DFS algorithm, and above all it can incorporate into the testing the use of sequences composed of the methods that belong to the API of the IoT.

3.1. Device A
The API of this equipment is composed of eleven methods, which were all described in specification API JSON file. Based on these methods and with a depth of two in the DFS algorithm the tool produced one hundred and thirty-two sequences. At the end of the testing process the tool discovered six minor issues on this equipment and produced the correspondent model. As mentioned before, a problem detected by the tool is anything that does not match the public description of the device API.

4. Conclusions
In this thesis we foresee that the IoTs will have a great impact in our world, controlling all sorts of devices on different aspects of our lives, from our work, to our personal lives, to our "society". For the present and for the future, we have to increase the testing of the software behind these devices, since the "virtual world" is having more and more impact in our physical world and personal lives and if we are to believe and trust the IoT’s with our personal information, then we need to guarantee a certain quality of the software that runs on these devices.

We started this thesis by studying the existing literature on API black-box testing. We realized that there were two major testing fields on how to test an API. The first field was fuzzing that allows to intensively test the input parameters of an API methods without having access to the source code. The other field was the test of how a sequence of method calls can influence the internal state of a device in order to show abnormal behavior. There was also a study of the bibliography on how to create a system model from the data accumulated from the tests, in other words, how we could create an accurate model that represents the internal model based on the inputs and outputs of the tests.

Only after the evaluation of the previous mentioned topics, we started the development of a new testing tool for the APIs of IoT devices that use REST protocol. The tool has three main goals: 1) to do fuzz testing on all the methods that belong to the API; 2) to test sequences that are combinations of these methods and; 3) to generate the model based on the results of these tests that represent sequences of the problems detected. For this testing to be possible the tool needs three files that describe the initial state of the IoT device, the configuration for the usage of the tool, and the methods available in the API together with all the security information necessary to produce the requests. The sequences are produced before the testing process starts and all the possible combinations are generated based on the level provided by the DFS algorithm that the tool uses to generate the sequences. After that the tool will automatically go through each one of these sequences testing the last method in the sequence. The sequence is used to exercise the testing of the methods in different internal states of the IoT device. During the fuzzing process the tool uses PyjFuzz, Radamsa and off_by_one tools to produce different request of the methods and the objective is to test the most diverse values for the different parameters present in the payload and also to test the url used by that method. After the testing process is finished we are left with three different kinds of logs. The detailed log that details various informations about the state of the IoT device and the calls that originated an error. The total log that contains all the calls made to the IoT device. The model log that
contains the sequences that had problems and that are used to produce the model. The model production starts after the testing process is finished, it uses the tool Synoptic and the model log to produce this model which together with the detailed log can show to the user different interactions between methods that have problems and help the user to understand how certain combinations of the problems may cause a catastrophic error in the IoT device.

We improved the state of the art in IoT testing, by creating a new tool that automatically tests the IoT devices through a combination of method sequence production with black-box testing, that will also generate the model that exhibits the problematic sequences of these IoT devices.

To evaluate the potential of this tool, we tested it in real scenarios with multiple IoTs that are already available in the market, in order to determine if the tool is capable of detecting errors/bugs or to detect abnormal behaviors in the IoT. In our tests we were able to detect several non conformance issues that contradicted the specification of the API of the devices being tested.

At the end, we accomplished to develop a tool, that is able to tackle some of the problems that exist on the other tools available in the world market. It fulfills all the objectives that we had set at the beginning: to be open source and based on tools that are also open source, to be intuitive, to do automatic testing with the minimum action of the user, to create a diversity of tests for each of the methods being tested, to produce a model of the sequences for the problems detected, and to be able to produce results (find problems) in IoT devices which are in production in the market. The only aspect that we weren’t able to fully fulfill, is the time that is necessary to produce the files that are used as
input by the tool. The tool uses these files to learn how to interact with the IoT, how to reset the device to the initial state, and to configure some parameters.

References
