Penetration Testing in the Context of Web of Things

TUM
Munich, Germany

***Abstract*—Penetration testing is an effective way of observing the security and safety levels of any given electronical system. A penetration test tries to gather information about access points of a given system which should not be reachable for the attacker. This paper focusses on penetration testing of IoT systems and methods used to exploit found vulnerabilities. It explains the method of fuzzing on the application layers of the IoT system as well as penetration testing tools used for each specific protocol. It also provides an overview of the IoT environment and lays out overarching concerns about security over all the protocols.**

***Index Terms*—IoT, Penetration Testing, Fuzzing, MQTT, HTTP**

I. Introduction

The Internet of Things (IoT) is the next step in the evolution of the Internet. It is an extension of the Internet, as a world wide network of servers and computers, by utilizing new capabilities of microcontroller with internet connections. It enables a broader range of devices to be able to communicate through the Internet and allows small devices to be integrated into steadily growing networks.

These devices vary drastically from their hardware to the resources they use as well as their different security implementations. Due to the great number of variations of these microcontrollers, these are a lot of different vulnerabilities that can be exploited. As these networks grow and become a vital part of the Internet the questions of security and safety arises. The computational limitations of microcontrollers lead to high demands on all layers of the devices and therefore need to be tested thoroughly. The paper highlights the growing necessity for better understanding the vulnerabilities of these microcontroller IoT systems. The growing number of these environment are build on only a few protocols but the high diversity of implementations introduced a lot of uncertainty regarding security and safety. Tools that help developers to find a basic common ground or help them to improve their systems through penetration testing are highly valuable.

Penetration tests try to probe IoT devices on different layers to find vulnerabilities. If such a vulnerability is found, an attack is started against the System under Test (SUT). This paper focusses on attacks on the application layer protocols like HTTP, MQTT and CoAP, which are commonly used between IoT devices. Furthermore we want to exclude Denial of Service (DoS) attacks as well as attacks against cloud platforms. We only considered device to device attacks that incorporated the use of valid protocols. These kinds of attacks try to interfere with messages of the used protocol to return data in a way that was not intended. One of these methods is called fuzzing, in which the valid protocol structure is emulated and (semi-)random data is send. This can lead to unintended actions of the SUT.

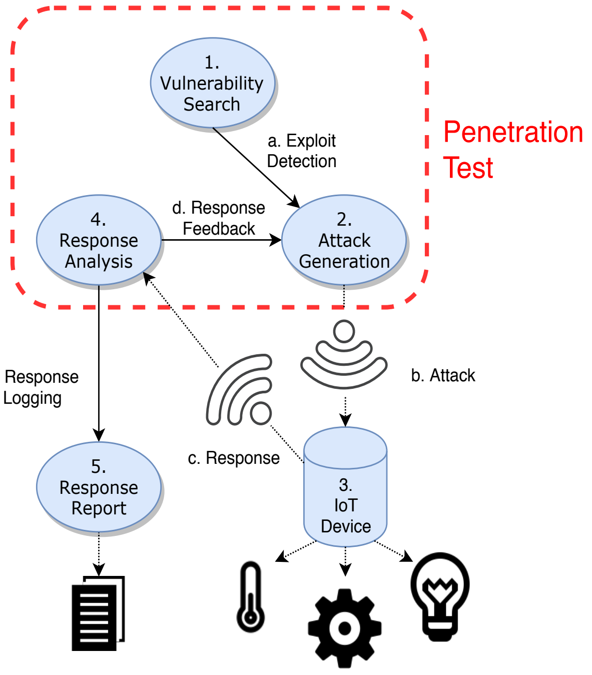

Fig. 1. Penetration Testing Tool

Figure 1 shows how such an attack schema should function. A penetration test starts with a search for vulnerabilities (1). The SUT (3) is scanned or probed and if vulnerabilities are detected (a), a fitting method of attack is selected (2). Which attack is valid highly depends on th SUT. If the communication is not encrypted and no authentication is needed, the attacker could start a fuzzing attack or sniff the system for crucial information (b). After the attack our test equipment records any responses of the SUT that might be returned (c). The response analysis (4) is the most important part of the testing system because is reveals how the SUT reacts to the inputs. All responses and information are logged (e) in a file (5) and a further attack is triggered (d) in accordance with the attack generation and newly collected information. Such a schema can be implemented manually by a human agent but also automated by using specialized tools for detected protocols.

In Section II, this paper describes methods called fuzzing to attack a SUT with different protocols. Section III focusses on automated tools used for penetration testing. Section IV sums up the paper in a conclusion and lays out points of interest for the future.

II. Fuzzing

Fuzzing [**?**] is an established attack on the application layer of devices. Hereby, the attacker uses the given protocol to communicate with the SUT. Fuzzing wants to explore the reactions of a SUT to a wide range of unpredictable inputs to observe possible flaws in the system as far as crashes of the device. The attacker inserts abnormalities into the communication and monitors the response.

There are deviations on how a fuzzing attack can be done. We differentiate between three types of fuzzers [**?**]:

- Random Fuzzer
- Mutative Fuzzer
- Generative Fuzzer

The first fuzzer randomly change parts of the communication as a brute force attack. The random fuzzer is easily implemented and covers a lot of possible inputs into the SUT but has a low chance of success if the SUT uses an underlying protcol which dismisses invalid inputs.

The mutative fuzzer limits its amount of randomness. It takes valid inputs of the SUT (either through recording/sniffing the SUT or through previous knowledge) and changes small portions of the message. This allows for a higher success rate than the random fuzzer but limits the coverage of possible inputs into the system.

The third type is the generative fuzzer. It incorporates methodical changes according to the underlying protocol that constitute as valid interactions with the SUT. This allows for targeted attacks with a low rejection rate of the messages.

Which fuzzer is needed, highly depends on the SUT in question. These fuzzers can have unpredictable outcomes, even with minor changes to the sent message, which makes a proper response logging extremely important. Fuzzing is a low effort attack and can be adjusted to the knowledge of the attacker about the SUT. If the system is a total black box with inputs and outputs, random fuzzing is an easy way to get reactions out of a SUT. If any background information exists, a mutative fuzzer or generative fuzzer might be harder to implement but could yield better results.

1) HTTP: The Hypertext Transfer Protocol (HTTP) is most commonly used for loading Hypertext documents form the Internet into a web browser but can also be used as a reliable protocol on the application layer for communication between devices in the same network. It is one of the most used protocols on the Internet as well as the Internet of Things.

Fuzzing [**?**], [**?**] on HTTP is done by altering packages in a way that they are still considered a valid request. This means while some data can be changed randomly, certain parts of the packages can not. It is possible to send random bytes of data in the body of an HTTP request but there are only a certain number of valid HTTP methods. Sensible modification of such requirements can lead to unexpected outcomes. Request methods or other protocol specific requirements are extremely important to take into account for HTTP because invalid request are immediately dismissed. There exist fuzzers that take preceding knowledge into account. Template based fuzzers and grammar based fuzzers [**?**] are two of these types. A template based fuzzer has knowledge about the basic structure of the protocol. It will produce test cases within the given outline but may produce a significant amount of invalid requests. A grammar based fuzzer has a certain vocabulary limited to valid inputs used by the system. Both of these fuzzers can be combined with random, mutative or generative fuzzers.

A good example for an HTTP fuzzer would be a grammar based generative fuzzer. It could incorporate the preceding knowledge about HTTP status codes with the limited grammar of HTTP requests like GET, PUT or POST. This kind of fuzzer would be able to create a high amount of valid HTTP requests/responses and systematically probe the SUT. A template based mutative fuzzer could be used a well but would yield a lower success rate of valid requests. Due to the strict HTTP protocol, a random fuzzer could only test if the overall protocol implementations was done properly on the SUT but would hardly produce any valid input.

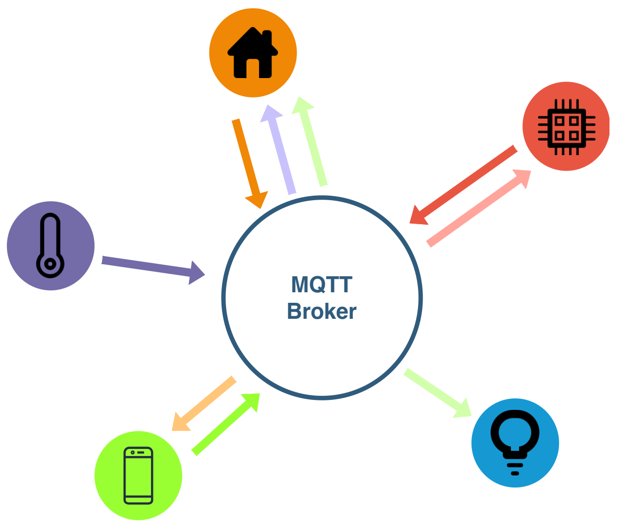

Fig. 2. MQTT Broker with Clients

2) MQTT: Message Queuing Telemetry Transport (MQTT) is a machine to machine protocol widely used between IoT devices. It is comprised of a star like structure with a server (Broker) in the middle and clients that connect to the broker. MQTT relies on a publish-subscribe structure [**?**]. A MQTT publish action is comprised of two parts, a topic and a message. The topic is the name of the queue in which the message ist published to the broker. The broker manages all interactions within the system. If one entity in the system publishes a message on a topic the broker will check if any entity might be subscribed to the given topic.

Figure 2 depicts such a MQTT broker/client system. It highlights the great level of centralization of MQTT systems. If the broker is not reachable, no communication will take place and the system is disabled. Because of this reason MQTT brokers are capable of handling a great number of clients and even more interactions. Another interesting fact about MQTT system can be taken from Figure 2. All communication is handled by the broker and it is the only entity in the system that knows about everything. This makes it the prime target if an attacker wants to take over the system. All other devices are sending and/or receiving data over the broker.

Fuzzing on MQTT is an effective way to test the protocol on a SUT. The MQTT protocol specification is public. This knowledge helps attackers to implement specialized fuzzers for the MQTT protocol. Due to the inherent publish/subscribe structure of MQTT every communication passes over the broker which makes direct targeting of devices (other than the broker) difficult without sniffing the system first.

TABLE I
MQTT HEADER BYTES (TABLE REWORKED FROM [?])

Bits	0	1	2	3	4	5	6	7
Byte1	Retain	QoS		DUP		Msg Bits		
Byte2	Remaining Length							

MQTT messages are comprised of at least two bytes which hold, as total minimum, 2 header bytes and no payload. These 2 bytes at the beginning of the message will always be present and hold information about, amongst other things, the kind of message send, a retain flag, the Quality of Service (QoS) level and the Duplication (DUP) flag. QoS has three level which determine if the message should be send at most once (level 0), at least once (level 1) or exactly once (level 2). The DUP flag is an indicator if the message was send before but did not receive an Acknowledgment. An MQTT fuzzer attack target these two header bytes because they are always present. Due to the knowledge about the exact positions of the header bits it is easier and more useful to use a template based fuzzer in comparison to a grammer based fuzzer. As we can se in Table 1, the first of the two bytes holds the most interesting bits can therefore be targeted easily with a prefabricated template. In this case random, mutative and generative fuzzers can be taken into account. A random fuzzer can test impossible header configurations like QoS level 0 and a set DUP flag. This configuration is impossible because a message can not be a duplicate if it is send at most once. A generative fuzzer could probe through different message types like subscribe or publish by utilising the knowledge about the public specification.

3) CoAP: Constrained Application Protocol (CoAP) is a protocol specially designed for IoT devices. Its has similarities to HTTP which make both protocols easily compatible. As a protocol designed for Embedded environments it was designed to support M2M communication an operates relatively resource friendly [?]. CoAP communication

Fuzzing on CoAP can be compared to both MQTT and HTTP. A grammar based fuzzer, explained in the HTTP section, would have the same advantage of selecting between valid request codes like GET, POST, PUT, DELETE for CoAP. This would create valid messages for testing the SUT.

Like MQTT, CoAP has a well defined structure for the first bytes. Instead of two bytes, CoAP utilizes 4 bytes with fixed locations for request/response codes and other header bits. This allows for the use of template based generative fuzzers [?].

III. PENETRATION TESTING TOOLS

A. MQTTSA

MQTT Security Assistant (MQTTSA) is a tool developed to improve the security awareness of developers using MQTT as the communication protocol for IoT systems [?]. According to research, some of the main causes of vulnerabilities in MQTT environments are the lack of security warnings as well as non secure default settings of the MQTT protocol. MQTTSA is capable of running DoS and fuzzing attacks as well as brute force attacks against the MQTT environment to breach authentication requirements and connect to the broker. If no authentication is needed or the tool breached the authentication it can operate as a template based fuzzer over a pre-programmed list of messages. Incorporated into MQTTSA is a sniffer to record messages send in the system. This can be achieved by subscribing to the ”#” topic which lets MQTTSA receive every message on any given topic.

Additionally to MQTTSA's capabilities to fuzz, sniff and act as a DoS attacker it allows the user to perform an active attempt of data tampering. This attack is not the same as fuzzing and a preceding sniff attack is crucial to extract any writeable topics of interest in the network.

After a successful testing of an IoT system, MQTTSA produces a fully automated report which includes all relevant information about all vulnerabilities tested and any breaches that may have been found. This makes MQTTSA a valuable and useful tool for MQTT security observation.

B. Burp Suite

Due to the fact that HTTP was first and foremost used as a protocol for the Internet and at a much later time became relevant for the IoT most tools for running penetration tests on HTTP are manly designed to test web applications. Therefore, specialized HTTP fuzzing tools are mostly embedded into larger framework of established existing programs. Burp Suite is such a powerful penetration testing tool [?]. It is capable of intercepting HTTP and HTTPS server-client communication but for the IoT Penetration testing only HTTP communication is of interest. This means it was not build for fuzzing attacks on IoT devices but nevertheless can be used as such a tool. It can generate valid HTTP requests and probe a SUT.

Within the Burp Suite framework an option called Burp Intruder allows to send brute force or fuzzing attacks. The attacker can select the address and port of the SUT and is able to send a payload. The tool allows the user to load customized files which could be prepared messages from a desired fuzzing architecture and send these messages to the

SUT. These messages have to be generated by another tool and can then be imported into Burp Suite as a file.

C. Applica

The following information is taken from [?]. It introduces a fuzzing tool called Applica which is structured around a customizable fuzzer for IoT systems embedded into a penetration testing framework. Applica is able to target HTTP, ZigBee and Bluetooth Low Energie (BLE). It is especially devised for testing IoT environments and therefore stands in contrast to Burp Suite as an HTTP tool. Applica analysis a network and is capable of extracting the package types send in the network (sniffing). After analysing the most common package type, it can infer the protocol. Applica then starts generating both valid and invalid messages over a fuzzing engine. This engine combines random, mutative and generative fuzzing technics to maximize the range of created messages. This method is, as we have seen in section II, a crude and basic attack with a low rate of success. To combat this problem, Applica combines the responses of the SUT after the first fuzzing attack with a another generative fuzzer to take promising messages that created a desired response and insert minor changes.

Applica is a powerful tool for IoT penetration testing and is one of the few tool that targets BLE and ZigBee protocols.

IV. Comparison

The penetration testing tools from section III are all powerful programs for attacking an SUT. After comparing them we can observe that they all incorporate similar technics to probe a system regardless of the protocol in use. All of them are able to sniff a system. This is a basic attack and crucial for probing application layer traffic. They area also capable to start fuzzing attacks as a powerful mean to actively attack on the application layer and interfere with the communication.

TABLE II
Comparison of Penetration Testing Tools

Features	Applica	Burp Suite	MQTTSA
MQTT	X	X	✓
HTTP	✓	✓	X
Other Protocol	✓	X	X
Integrated Msg Generator	✓	X	✓
Create Report File	✓	X	✓
Load custom Msg File	X	✓	X

As shown in Table II, the three penetration testing tool differ not only in their target protocols but also in other important dimensions. On of the most important parts of fuzzing is generating desired test cases. Applica and MQTTSA have integrated fuzzer architectures that can be modified to generate test messages. Burp Suite does not have such an option. It has to import a file that must be generated by an external fuzzer. While this fact could be seen as negative, it also allows Burp Suite to combine any kind and amount of fuzzing architectures openly available. The ability to load fully customized files is therefore only given by Burp Suite. Applica and MQTTSA allow templates to be loaded, which limits their flexibility. This is one of the prime examples where one can see that Burp Suite was not directly designed for testing IoT environments.

Tracking the outcome of the attack messages is extremely important to recreate desired outcomes. Applica and MQTTSA both return an attack report at the end of every automated test session. They track all attacks and list any vulnerabilities found. Burp Suite does not provide this option. It only returns a log file with all interactions and successful attacks have to be determined manually or through another software.

Applica is the only penetration testing tool that works with more than one protocol. It is not only able to attack MQTT environments but also works on BLE and ZigBee communication. It is an attempt to unify different IoT systems under one testing tool.

V. Conclusion

To Conclude, penetration testing of IoT environments is a crucial part of the development of these systems and tools like MQTTSA or Applica are the beginning of a growing awareness of security and safety for the Internet of Things.

References

[1] D. Serpanos, and M. Wolf, Internet-of-Things (IoT) Systems, Springer, 2018

[2] R. McNally, K. Yiu, D. Grove, and D. Gerhardy, “Fuzzing: the state of the art”

[3] V. Sachidananda, S. Bhairav, N. Ghosh, and Y. Elovici, ”Pit: a probe into internet of things by security analysis”

[4] ”MQ Telemetry Transport” http://mqtt.org

[5] S. Hernandez Ramos, M. T. Villalba, and L. Lacuesta, ”MQTT security: a novel fuzzing approach”

[6] R. A. Rhaman, and B. Shah “Security analysis of IoT protocols: a focus in coap,” 2016 3rd MEC International Conference on Big Data and Smart City.

[7] B. S. Melo, P. L. de Geus, and A. A. Gregio “Robustness testing of coap server-side implementations through black-box fuzzing techniques,”

[8] A. Palmieri, P. Prem, S. Ranise, U. Morelli, and T. Ahmad, “MQTTSA: a tool for automatically the secure deployments of mqtt brokers,” 2019 IEEE World Congress on Services (SERVICES)

[9] S. Wear, The Burp Suite Cookbook, Packt Publishing Ltd, 2018.