Towards a Mutation Analysis of IoT Protocols

Benjamin Aziz

School of Computing, University of Portsmouth, Portsmouth, United Kingdom

Abstract

Context: Mutation testing and analysis is concerned with the introduction of single faults (or errors) into a system's design, specification, implementation or interface and then testing or analysing the effects caused by those faults or errors on the system's properties and behaviour. Such faulty entities are called mutants. **Objective:** This short paper sketches the idea that mutations can be used to identify whether protocol implementations may deviate from the standards defining those protocols. **Method:** We apply formal analysis techniques to analyse the harmful effects of mutations on IoT protocol specifications, most importantly, e.g. whether those mutations will violate the IoT protocol standard. **Results:** We discovered in our initial investigation one interesting case where a mutant protocol specification may lead to implementations that drop every message intended for publication to applications without breaking the standard of quality of message delivery in the protocol standard. **Conclusion:** We believe as a result, that there is some additional investigation needed in this direction that could be beneficial in implementing future more reliable protocols and also that the current standards need to be revised to take care of such scenarios.

1. INTRODUCTION

Mutation testing and analysis is concerned with the introduction of single faults (or errors) into a system's design, specification, implementation or interface and then testing or analysing the effects caused by those faults or errors on the system's properties and behaviour. Such faulty entities are called *mutants*. Despite the fact that mutation testing and analysis has a wide scope of research [1], its application using formal specification and verification techniques has remained somehow limited over the years.

In this short paper, we consider the general idea that mutations could be used as a technique in formally understanding whether faulty implementations of systems can pass as correct implementations in relation to the standards defining those systems. In particular, we take the example of an Internet of Things (IoT) protocol, namely the MQTT standard [2], and show that such cases of faulty but nonetheless standard-compliant specifications can indeed exist. We only demonstrate this approach to one case of MQTT, when the Quality of Service

(QoS) level of message delivery is set to 0. Unlike testing, which is incomplete, we advocate the use of formal static analysis approaches (e.g. [3]) in order to verify the effects mutants will have on the protocol's message delivery semantics. We have already shown in [4] how formal techniques can be useful for analysing IoT protocols.

The idea of applying mutations to protocols has its roots in works such as [5, 6]. In [5], the authors introduced the idea of mutating the nature of a message based on the assumption that network interference is the only source of faults. On the other hand, [6] adopt a more general approach based on a set of common mutations that can occur during the specification of cryptographic protocols. The type of mutations and the formal language used in [5, 6] are different from those used in this paper. Our treatment here is motivated by mutations that can impact the general behaviour of a system. We also advocate a more rigorous method using process algebraic specifications in order to remove any ambiguity about the intended behaviour.

2. Specifying IoT Protocols

The formal language we use here to model IoT protocols is the π-calculus process algebra [7], which has the following syntax defining processes $P, Q \in \mathcal{P}$ based on names $x, y \in \mathcal{N}$:

$$P, Q ::= \overline{x}\langle y\rangle.P \mid x(y).P \mid \;!P \;\mid\; (\nu x)P \;\mid\; (P \,|\, Q) \;\mid\; (P{+}Q) \;\mid\; \mathbf{0} \;\mid\; A(x)$$

The input actions, $x(y).P$, synchronises with suitable output actions $\overline{x}\langle y\rangle.P$. The other constructs include process replication $!P$, new name creation $(\nu x)P$, parallel composition $(P \mid Q)$, non-deterministic choice $(P + Q)$ and the null process $\mathbf{0}$. Finally, we utilise *process definition calls* in the form of $A(x)$ to pass the value of some input x to a process A defined as $A(y) \stackrel{\text{def}}{=} P$, where x will replace y in P. If the definition A does not accept any input parameters, then we simply omit the input parameter y and write $A() \stackrel{\text{def}}{=} P$.

We give here one example for the model of the MQTT QoS=0 protocol [2]:

$$\begin{aligned}&\mathit{Protocol}() \stackrel{\text{def}}{=} (!((\nu \mathit{Publish})\,\mathit{Client}(\mathit{Publish}))) \mid !\mathit{Server}() \qquad \text{where,}\\ &\mathit{Client}(z) \stackrel{\text{def}}{=} \overline{c}\langle z\rangle.\mathbf{0} \qquad \text{and} \qquad \mathit{Server}() \stackrel{\text{def}}{=} c(x).\overline{\mathit{pub}}\langle x\rangle.\mathbf{0}\end{aligned}$$

In this protocol, the Client continuously creates new messages (*Publish*) (e.g. related to some topic of interest) that it outputs on a channel c, which is then received at the Server. The Server, in turn, outputs this message on another channel called *pub* on which a Subscriber is listening.

Our model of the subscribers is minimal where the Subscriber simply only (repeatedly) consumes the published messages after which it terminates:

$$\mathit{Subscriber}() \stackrel{\text{def}}{=} \;!(\mathit{pub}(x').\mathbf{0})$$

It is worth noting here that publish/subscribe systems have been modelled in the past using process algebra, e.g. as in [8].

3. Generating Mutations

Mutations can include various types of alterations to the syntax of processes, but for brevity, we give here only three examples of such alterations.

3.1. Free Name Changes

These include alterations of a single free name in a process. For example, this would result in the Client process above mutating to $\overline{d}\langle z\rangle.\mathbf{0}$ and $\overline{c}\langle q\rangle.\mathbf{0}$, and the Server process to $d(x).\overline{pub}\langle x\rangle.\mathbf{0}$ and $c(x).\overline{pub'}\langle x\rangle.\mathbf{0}$, where $pub \neq pub'$. Such alterations may be due to network interference changing messages' content or bad implementations changing communication channel addresses.

3.2. Action Changes

Another kind of mutations that one may consider are modifications to single actions in a process. Such alterations can in reality represent the wrong choice or misconfiguration of a device or a software component, where the behaviour (input/output) is not as expected in the specification. In our example, the Client process could be changed to the mutants $c(y).\mathbf{0}$, $d(y).\mathbf{0}$ and $\overline{d}\langle q\rangle.\mathbf{0}$, and the Server process changed to $\overline{d}\langle q\rangle.\overline{pub}\langle x\rangle.\mathbf{0}$, $\overline{c}\langle q\rangle.\overline{pub}\langle x\rangle.\mathbf{0}$, $c(x).pub(y).\mathbf{0}$ and $c(x).pub'(y).\mathbf{0}$. As discussed in the next section, the third client mutant, $\overline{d}\langle q\rangle.\mathbf{0}$, is an interesting case as it can cause data to be sent to the wrong server.

3.3. Process Operator Changes

These mutations include single changes to process operators, which can express bad compositions of the various components of a system's implementation. For the Client process, this would result in the mutants $\overline{c}\langle z\rangle+\mathbf{0}$ and $\overline{c}\langle z\rangle \mid \mathbf{0}$, and for the Server process, the mutants $(c(x).\mathbf{0}) + (\overline{pub}\langle x\rangle.\mathbf{0})$, $(c(x).\mathbf{0}) \mid (\overline{pub}\langle x\rangle.\mathbf{0})$, $c(x).((\overline{pub}\langle x\rangle.\mathbf{0})+\mathbf{0})$ and $c(x).((\overline{pub}\langle x\rangle.\mathbf{0}) \mid \mathbf{0})$. As we discuss in the next section, this last mutation has a denial-of-service impact on the subscribers expecting to obtain data from the server.

4. Initial Findings and Conclusion

As an initial investigation, we applied formal static analysis techniques, developed in [3] to capture message substitution in communicating processes, to analyse harmful effects of our mutations on the MQTT clients and servers. Our approach considers a mutant to be *harmful* if it causes unsafe/insecure behaviour in the protocol, which does not violate the protocol's standard QoS semantics and hence the mutant can be implemented without any issues.

We can summarise our initial findings as follows:

- **Case of free name changes**. These changes produced some potentially dangerous mutants such as $c(x).\overline{pub'}\langle x\rangle.\mathbf{0}$, where the Server outputs the message to the wrong Subscriber, and $d(x).\overline{pub}\langle x\rangle.\mathbf{0}$, where the message is inputted from the wrong Client. However, such mutations violate the MQTT definition of communications and can therefore be prevented in MQTT implementations.

- **Case of action changes**. The main mutant here was the client process $\overline{d}\langle q\rangle.\mathbf{0}$, which has the potential of outputting some data q to the wrong server communicating over channel d. This is dangerous since the server listening on d may be unauthorised, and if q is some sensitive internal data that must not be revealed (e.g. an encryption key). Again, such mutations violate the protocol's specification and are therefore preventable in implementations.

- **Case of process operator changes**. Our major finding was in the case of the mutant $c(x).(\overline{pub}\langle x\rangle.\mathbf{0} + \mathbf{0})$, where we find that the Server process has a choice between publishing the inputted message to the Subscriber or doing nothing. Worse, it may always wish to opt for $\mathbf{0}$ ignoring to ever publish any messages. This behaviour does not violate the MQTT QoS=0 quality standard since the mutant Server process turns itself into a process that "simulates" a lossy network, which is behaviour that QoS=0 is willing to tolerate. We consider such a mutant to be implementing a form of denial-of-service, since it is conceptually a *server* and not a *network*, and therefore it should attempt to deliver any data generated by the clients. The same issue is implicitly present in $c(x).(\overline{pub}\langle x\rangle.\mathbf{0} \mid \mathbf{0})$ since parallelism can be interpreted with non-deterministic choice as was shown in [9].

For future work, we plan to develop the algorithms underlying the mutations above and to apply these along with the static analysis techniques of [3] on the other two MQTT protocols, namely QoS=1 and QoS=2, in order to provide a complete study. We also plan to apply our mutation analysis approach to other communication protocols, including other IoT protocols. Finally, there is currently no automatic way in comparing the traces of a mutated process with the original one in order to reason on whether the former is violating the latter. We plan to investigate this direction further, by formalising a trace refinement relationship akin to that introduced in [10].

References

[1] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Transactions on Software Engineering 37 (5) (2011) 649–678.

[2] A. Banks, R. Gupta, MQTT Version 3.1.1 Plus Errata 01 (2015).

[3] B. Aziz, G. Hamilton, D. Gray, A static analysis of cryptographic processes: The denotational approach, Journal of Logic and Algebraic Programming 64(2) (2005) 285–320.

[4] B. Aziz, A Formal Model and Analysis of an IoT Protocol, Ad Hoc Networks 36 (P1) (2016) 49–57.

[5] J. Jürjens, G. Wimmel, Formally Testing Fail-Safety of Electronic Purse Protocols, in: Proceedings of the 16th IEEE International Conference on Automated Software Engineering, ASE '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 408–.

[6] F. Dadeau, P. Héam, R. Kheddam, Mutation-Based Test Generation from Security Protocols in HLPSL, in: Fourth IEEE International Conference on Software Testing, Verification and Validation, ICST 2011, Berlin, Germany, March 21-25, 2011, IEEE Computer Society, 2011, pp. 240–248.

[7] R. Milner, J. Parrow, D. Walker, A Calculus of Mobile Processes, Information and Computation 100(1) (1992) 1–77.

[8] F. Siewe, H. Zedan, A. Cau, The calculus of context-aware ambients, Journal of Computer and System Sciences 77 (4) (2011) 597 – 620.

[9] I. Stark, A fully abstract domain model for the π-calculus, in: Proc. of the 11^{th} Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society, New Brunswick, New Jersey, USA, 1996, pp. 36–42.

[10] M. Gieseking, C. von Ossietzky, Trace Refinement of π-Calculus Processes, `https://www.uni-oldenburg.de/fileadmin/user_upload/informatik/ag/csd/gieseking15_ffm.pdf`, accessed: 18-04-2018.