Enhancing Transparency of MQTT Brokers For IoT Applications Through Provenance Streams

Milan Markovic
milan.markovic@abdn.ac.uk
Computing Science,University of Aberdeen
UK

Peter Edwards
p.edwards@abdn.ac.uk
Computing Science,University of Aberdeen
UK

ABSTRACT

Systems based on Internet of Things (IoT) technologies may violate user privacy if personal data they produce and use become available to unauthorised agents. Recording provenance of IoT system behaviour may support assessment mechanisms ensuring compliance of system components with data access constraints. In this paper, we describe a prototype implementation of a provenance-enabled MQTT broker enhanced with the ability to generate provenance records describing the actual broker behaviour during message forwarding. The implementation utilises a semantic stream based approach for generating and analysing provenance data to discover message forwarding to untrusted agents. The initial evaluation demonstrates the feasibility of semantic solutions in this context.

KEYWORDS

internet of things, provenance, transparency, mqtt

1 INTRODUCTION

IoT infrastructures consist of a number of physical and virtual devices networked using a range of technologies.

The Message Queuing Telemetry Transport (MQTT) protocol [1] is a lightweight publish/subscribe messaging protocol for M2M communication. MQTT brokers are frequently used as an integral part of IoT infrastructures to provide a communication bridge between different system components [4, 9].

We argue that ensuring transparency of such brokers is a critical enabler for assessing various aspects of IoT systems such as accountability, privacy, quality and security. In this context, transparency may include information on various aspects of the deployed system such as broker configuration and the location where data processing takes place; however, it may also include information regarding data flows and agents that have access to the processed data.

For example, sensors deployed as part of an IoT system may be collecting personal data such as users' location, health information, etc. In order to ensure privacy in such settings, it is important that such data is not made available to unauthorised agents. In the MQTT context, this means that the broker should not forward messages received under a certain topic to unauthorised subscribers. To achieve this, brokers may be extended with various authorisation mechanisms [8], which are not part of the core MQTT specification and hence not always implemented. If a broker was deployed without such a protection layer, or the layer malfunctioned, the data breach could be detected from logs generated during broker operation (e.g. information including timestamps, message payloads, clients to which the message was forwarded, etc.). However, such logs often exist as text files without any formally defined structure, which makes their processing difficult. We argue that to enhance the utility of such records, and the overall transparency of MQTT brokers, these logs should be captured in a machine-understandable format to enable intelligent audit processes to assess whether a broker is behaving as expected.

In our prototype solution presented in this paper, we apply a provenance-based approach to modelling of data describing an MQTT broker's operation during message forwarding. This produces provenance graphs that include entities (e.g. message topics), agents (e.g. subscribers receiving messages), and activities (e.g. message forwarding) as nodes and relationships between these concepts as connecting edges. In our prototype, we also implement and evaluate the linked data stream approach [2] for creating provenance streams that enable on-the-fly event detection without the need for storage of elaborate logging data which could pose scalability issues.

2 BACKGROUND

2.1 The MQTT Protocol

The MQTT protocol (version 3.1.1) is an OASIS standard[1] publish/subscribe messaging transport protocol for lightweight client-server communication in IoT infrastructures. The protocol is typically run over TCP/IP networks with small transport overhead and three basic message delivery quality assurances including the following settings: message delivered at most once (*QoS 0*); at least once (*QoS 1*); and exactly once (*QoS 2*). The protocol does not impose any restrictions on the structure of the message payload.

2.2 MQTT-Plan, EP-Plan and PROV-O

Ontologies define types of concepts and relationships between them that can be used to build semantic graphs. The graphs consist of triples where each triple has three parts: subject, predicate, and object. Each part of a triple is associated with a unique URI[2]. Graphs can be queried by SPARQL[3] queries returning instances that match query patterns (Figure 1).

M4IoT '19, December 9–13, 2019, Davis, CA, USA

ACM ISBN 978-1-4503-7028-8/19/12…$15.00
https://doi.org/10.1145/3366610.3368099

[1] http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
[2] A prefix separated by : abstracts the repetitive parts of the URI. For example, ex: can represent http://example.com#
[3] https://www.w3.org/TR/sparql11-query/

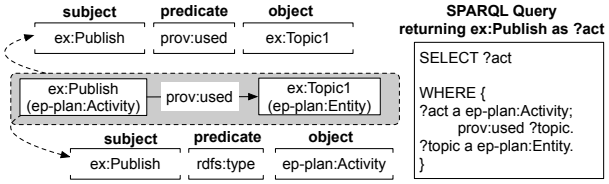

Figure 1: An overview of semantic technologies for representing and querying graph data.

The MQTT-Plan [6] ontology defines a semantic vocabulary for describing high level abstract plans associated with MQTT brokers and their corresponding execution traces. The ontology captures information that could be found by inspecting individual MQTT control packets (e.g., message topics) and other information maintained by the broker (e.g. the identity of publishers and subscribers). MQTT-Plan extends P-Plan [3], which enables descriptions of plans as acyclic graphs in the form of a series of steps and variables representing their inputs and outputs with links to the corresponding parts of execution traces. MQTT-Plan also extends PROV-O [5], a W3C Recommendation, which is used to record the execution trace (i.e. the actual behaviour) as a collection of entities used and produced by activities, as well as agents that bear some responsibility for those activities.

The EP-Plan ontology [7] is a recent extension of P-Plan (fully compatible with MQTT-Plan) which enables additional metadata (e.g., constraints) to be associated with plan descriptions and provides enhanced mechanisms for integrating provenance records generated by different components (e.g. brokers, sensors, etc.) of a complex IoT system.

2.3 C-SPARQL framework

The *C-SPARQL framework*[4] utilises the C-SPARQL extension of SPARQL (a query language for querying semantic graphs) to construct continuous queries on streams of linked data [2]. The framework integrates relational stream processing framework ESPER[5] with a SPARQL engine to associate timestamps with individual triples (i.e. individual statements captured in an RDF[6] graph), which are transformed into so-called quadruples. By extending SPARQL with the concepts *stream* and *window*, continuous SPARQL queries can be used to retrieve triples within some time range (i.e. a window) from a specific quadruple sequence. The *C-SPARQL framework* provides methods to query only the most recent windows which can be determined either by specific time (e.g. last 10 seconds) or specific number of triples flowing through the stream (e.g. last 1000 triples).

3 PROVENANCE ENHANCED MOQUETTE

We have extended the opensource Moquette MQTT broker[7] with a set of classes integrating the C-SPARQL framework for processing semantic streams and the JENA framework[8] for creating semantic provenance graphs. Our implementation is available as an open source project[9].

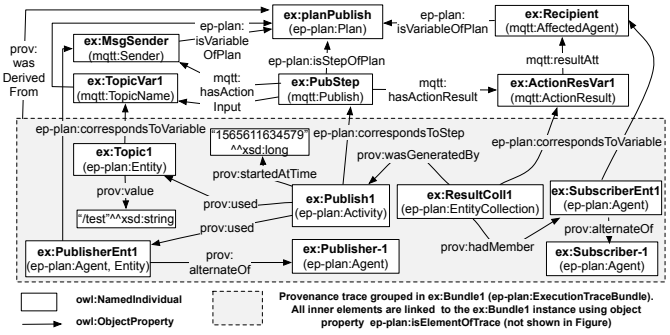

Figure 2: A provenance trace documenting message republishing with links to the corresponding plan description.

```
REGISTER QUERY myquery AS
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX ep-plan: <https://w3id.org/ep-plan#>
PREFIX tl: <https://trustlens.org#>
PREFIX ex: <http://example.com/mqtt-plan#>
Construct {
 ?elementOfTrace ?s ?o.
 ?bundle ?x ?y.
 }
FROM STREAM <http://iotstreams> [RANGE TRIPLES 81000]
FROM <https://trustlens.org/trustedAgents>
WHERE {
   ?agent  ep-plan:isElementOfTrace ?bundle;
           ep-plan:correspondsToVariable ex:AffectedAgent.
   ?activity prov:used ?topic; ep-plan:isElementOfTrace ?bundle.
   ?result prov:wasGeneratedBy ?activity; ep-plan:isElementOfTrace ?bundle;
           prov:hadMember ?agent.
   ?topic  ep-plan:isElementOfTrace ?bundle; prov:value "/test";
           ep-plan:correspondsToVariable ex:Topic.
   ?elementOfTrace ep-plan:isElementOfTrace ?bundle; ?s ?o.
   ?bundle ?x ?y.
FILTER NOT EXISTS {?agent a tl:TrustedAgent.}
}
```

Listing 1: A C-SPARQL query used to examine the provenance stream.

Figure 2 illustrates a provenance trace generated by a broker using the MQTT-Plan, EP-Plan and PROV-O vocabularies and its links to the corresponding plan description. To keep the integration lightweight we omitted entities from the provenance record that represent message payloads and control packets, as these were not required by our use case for detecting unauthorised message recipients. At the core of the provenance trace is the *ep-plan:Activity* which represents a single execution of the *mqtt:Publish step*. This is then linked to *ep-plan:Entities* representing individual topics and sender identities used to guide the publish process, as well as the identity of an agent to which a message has been forwarded. Each element of the execution trace is linked to a unique *ep-plan:ExecutionTraceBundle*, which is a utility concept to support querying of multiple provenance traces within a single graph.

Every time a message is republished by the broker, a new provenance trace is constructed using the MQTT-Plan, EP-Plan and PROV-O vocabularies and is entered on the linked data stream. A continuous C-SPARQL query (e.g., Listing 1) is then used to

[4]http://streamreasoning.org/resources/c-sparql
[5]http://www.espertech.com/esper/
[6]https://www.w3.org/RDF/
[7]https://github.com/moquette-io/moquette
[8]https://jena.apache.org
[9]https://github.com/TrustLens/moquette-prov

identify provenance information that documents non-compliance (i.e. a message was forwarded to an unauthorised client) which is then saved locally as a TTL[10] file. The stored provenance graphs documenting non-compliance can then be queried for relevant information, using a query of the kind presented in Listing 2.

```
PREFIX ep-plan: <https://w3id.org/ep-plan#>
PREFIX tl: <https://trustlens.org#>
PREFIX ex: <http://example.com/mqtt-plan#>
PREFIX mqtt: <http://w3id.org/mqtt-plan#>
PREFIX prov: <http://www.w3.org/ns/prov#>

SELECT Distinct ?sender ?recipient ?time
WHERE {
?senderVar a mqtt:Sender.
?senderEntity ep-plan:correspondsToVariable  ?senderVar;
              prov:alternateOf ?sender.
?activity prov:used ?senderEntity;prov:startedAtTime ?time.
?result prov:wasGeneratedBy ?activity; prov:hadMember ?recipiententity.
?recipiententity prov:alternateOf ?recipient;
                 ep-plan:correspondsToVariable ?recipientVar.
?recipientVar a mqtt:AffectedAgent.}
```

Listing 2: An example SPARQL query for inspecting provenance traces.

4 EVALUATION

In this section we describe the results of our initial evaluation of the prototype implementation. The evaluation focused on the effects of provenance mechanisms on the latency of delivered messages and time required to process semantic streams under different scenarios such as varying numbers of triples per processing window, number of messages forwarded, etc.

4.1 Set up

The evaluation was performed on a virtual server instance running Ubuntu 16.04.5 LTS with 72GB EDO DIMM and 4 CPU cores Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz. The same server instance was used to run the MQTT broker (started with 6GB of heap size) and the pool of MQTT clients[11] for publishing/receiving messages, which were implemented using the PAHO Java library[12]. The stream engine used the C-SPARQL query shown in Listing 1 to monitor the provenance stream. The content of each stream window containing the provenance traces was merged with the static knowledge-base which was loaded using the C-SPARQL engine API during the broker's startup process. The static knowledge contained the identities of trusted agents (see Listing 3).

```
<https://trustlens.org#sub-461>  a  <https://trustlens.org#TrustedAgent> .
<https://trustlens.org#sub-46> a  <https://trustlens.org#TrustedAgent> .
```

Listing 3: A portion of static knowledge encoded in the TTL format. Each instance representing the ID of a known subscriber is assigned the type TrustedAgent.

Since the stream windows that the query was evaluating contained more than one provenance trace (with all elements contained within an *ep-plan:ExecutionBundle*), the query identified bundles that contained a record of the topic we were interested in observing (e.g. "/test"). Using SPARQL's filtering function, the query also selected all triples relating to bundles which contained a record of an agent receiving a published message, and whose identity was not found in the static knowledge base of trusted agents.

If such triples were found, they were exported as a graph into a local file. This graph then represented a provenance trace documenting a non-compliance episode. No other provenance traces were stored by the system.

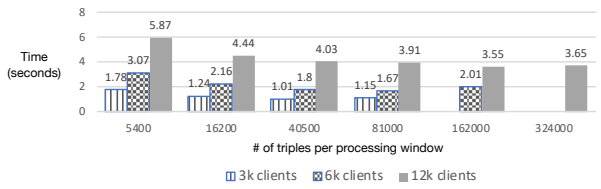

Figure 3: Time required to process a stream of provenance traces with varying processing window sizes against different volumes of subscribers.

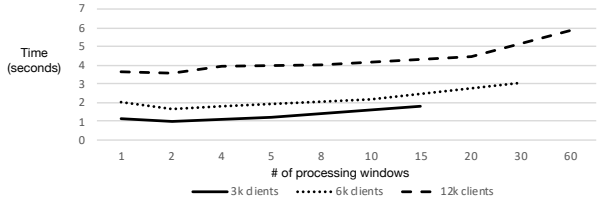

Figure 4: Time required to process a stream of provenance traces with varying numbers of processing windows against different volumes of subscribers.

4.2 Results

Experiment 1 focused on identifying optimal sizes of processing windows (i.e. number of triples per window). We created a pool of 3k, 6k and 12k subscribers which were listening to the topic "/test" and one client publishing messages under that topic with QoS 0. We then varied the number of triples per processing window and measured the time required to process all provenance traces generated when a message was republished to all subscribers. For each setting, four messages were sent during the warm up stage. This was followed by three additional messages during which we measured the time required to process the full provenance stream. We then averaged the time from these three observations (the same approach was applied to other experiments described in this section). Figure 3 and 4 show that the system performed best when the processing load was split in 1-2 processing windows[13]. As illustrated by Figure 4 there seems to be a small saving if the overall processing load is split into two windows.

Experiment 2 evaluated whether our implementation of provenance mechanisms had any significant impact on the baseline performance of the Moquette broker. We created different republishing loads on the broker and measured the latency at which the messages were received by clients. In this experiment we again created

[10] https://www.w3.org/TR/turtle/

[11] We experienced issues when generating more than 600 clients in a single Java application. As a result, the pool of MQTT clients was created by executing multiple jar files each creating 600 clients.

[12] https://www.eclipse.org/paho/

[13] Note that with 3k subscribers and window size set to 81k triples the system only processed one window (i.e. the size of a single provenance trace describing message forwarding to one subscriber is 27 triples)

a network of a single publisher publishing under a single topic to a pool of subscribers. Each message was associated with a timestamp, which was generated when the message was sent, and each client then calculated the latency based on the time when the message was received. Table 1 shows a very small increase of 40 - 50 ms in average message latency for 3k and 6k subscribers.

	Number of subscribers	1200	3000	6000
Without provenance	MIN	29	28	36
	MAX	1039	1064	1105
	AVG	532	545	552
With provenance	MIN	35	31	35
	MAX	1045	1106	1119
	AVG	531	588	603

Table 1: Observed message latency in ms.

In *Experiment 1*, for every test we loaded a static knowledge base containing 12k of triples and each test generated one provenance trace of non-compliance identifying a single untrusted agent. In *Experiment 3*, we were interested in evaluating whether the size of the static knowledge base, and varying populations of trusted and untrusted agents present in the stream had any impact on processing time. We created a pool of 6k subscribers and set the window size to 81k triples (based on the results from *Experiment 1*). We then varied the size of the static knowledge containing information about trusted agents and the populations of untrusted agents receiving messages.

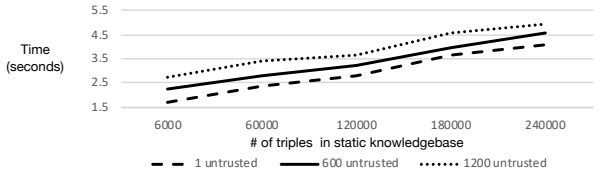

Figure 5: Time required to process a stream of provenance traces with varying numbers of untrusted agents and varying static knowledgebase size.

Figure 5 shows the results, indicating that increasing the size of the static knowledge base indeed has a negative impact on the time required to process the provenance stream. However, the function of the growth of processing time appears to be free of any exponential increases that could hinder the scalability of the solution. Figure 5 also shows that the processing time is impacted by the number of untrusted agents that are detected. This could be explained by the higher demand for resources from file writing processes that have to write larger files describing saved provenance traces. In future, this could perhaps be improved by writing into an in-memory triple store rather than onto a disk.

4.3 Limitations

Currently, the proposed approach has a number of limitations. For example, it assumes that information captured in the static knowledge base (i.e. the list of trusted agents) can be linked to the ID of the MQTT client. An alternative approach could be to calculate the trustworthiness of a client on the fly based on some characteristics included in the provenance trace that can be observed by the broker (e.g. sender's location).

Another limitation is related to the case where the size of the stream processing window is larger than the sum of the triples contained by the generated provenance traces. Such a window would not be evaluated until additional provenance traces had filled the required window size. This could be mitigated by altering the C-SPARQL engine implementation to force such windows to be evaluated after a certain time period (e.g. 500ms).

Finally, using the same server instance to run the broker and the client pool would mean more CPU demand which could have an effect on the observed performance results.

5 CONCLUSIONS & FUTURE WORK

In this paper, we presented a partial implementation of a semantic provenance-based transparency mechanism within a third party MQTT broker. Our initial evaluation results suggest that implementing semantic solutions within middleware IoT components offers a feasible and potentially scalable approach to enhance their auditability.

In our future work, we will focus on extending the provenance mechanisms to cover other parts of a broker's behaviour (e.g. handling connection requests) as well as updating the MQTT-Plan ontology to align with the recently introduced version 5 of the MQTT protocol[14]. We will also explore the opportunities for recording additional plan metadata provided by the EP-Plan ontology and integration of MQTT provenance in the wider IoT system context. We also plan to test the separation of the stream processing engine from the broker in a distributed network set up.

ACKNOWLEDGMENTS

The work described here was funded by the award made by the RCUK Digital Economy programme to the University of Aberdeen (EP/N028074/1).

REFERENCES

[1] Banks, A., and Gupta, R. Mqtt version 3.1.1. Tech. rep., OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, 29 October 2014. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[2] Barbieri, D. F., Braga, D., Ceri, S., Valle, D. E., and Grossniklaus, M. Querying rdf streams with c-sparql. *SIGMOD Rec. 39*, 1 (2010).

[3] Garijo, D., and Gil, Y. Augmenting prov with plans in p-plan: scientific processes as linked data. CEUR Workshop Proceedings.

[4] Lampkin, V., Leong, W. T., Olivera, L., Rawat, S., Subrahmanyam, N., Xiang, R., Kallas, G., Krishna, N., Fassmann, S., Keen, M., et al. *Building smarter planet solutions with mqtt and ibm websphere mq telemetry.* IBM Redbooks, 2012.

[5] Lebo, T., Sahoo, S., and McGuinness, D. PROV-O: The PROV ontology. Tech. rep., April 2013.

[6] Markovic, M., Corsar, D., Asif, W., Edwards, P., and Rajarajan, M. Towards transparency of iot message brokers. In *International Provenance and Annotation Workshop* (2018), Springer, pp. 200–203.

[7] Markovic, M., Garijo, D., Edwars, P., and Vasconcelos, W. Semantic modelling of plans and execution traces for enhancing transparency of iot systems. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)* (2019), IEEE.

[8] Niruntasukrat, A., Issariyapat, C., Pongpaibool, P., Meesublak, K., Aiumsupucgul, P., and Panya, A. Authorization mechanism for mqtt-based internet of things. In *2016 IEEE International Conference on Communications Workshops (ICC)* (2016), IEEE, pp. 290–295.

[9] Soni, D., and Makwana, A. A survey on mqtt: a protocol of internet of things (iot). In *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)* (2017).

[14] https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html