

Analyzing MQTT Attack Scenarios: A Systematic Formalization and TLC Model Checker Simulation

Amina Jandoubi¹^a, M. Taha Bennani¹^b, Olfa Mosbahi²^c and Abdelaziz El Fazziki³^d

¹LIPSIC Laboratory, Faculty of Sciences of Tunis, University of Tunis El Manar,
Tunis, 2092, Tunisia

²LISI Laboratory, National Institute of Applied Sciences and Technology (INSAT), University of Carthage,
Tunis, 1080, Tunisia

³Computer Science Dept, LISI Laboratory, Caddi Ayyad University of Marrakesh, Marrakech, Morocco


Keywords: SIGIRO, MQTT, LTL, Formalizing, Attack Scenarios, TLC Model Checker.


Abstract: The SIGIRO project seeks to create an intelligent system for managing water resources in Marrakech-Safi and Tunisia's northwest regions. The project introduces a systematic monitoring process to ensure adaptive control to address climate change. SIGIRO gathers data using the MQTT protocol, which has been the target of several cyberattacks in recent years. The absence of a formal description of these attacks leaves the field open to interpretation, leading to distinct implementations for a given attack. In this article, we formalize these attacks, provide descriptions, and check their exactness. We offer a systematic approach to formalizing seven attack scenarios targeting the MQTT protocol. Using the LTL temporal logic formalism, we generate 12 LTL formulas, each precisely describing a specific attack scenario. We classify these formulas into four categories according to a sequence of observation and injection events. These events are the abstract elements needed to control the attacks' implementation. We verify our proposed formulas using the TLC Model Checker. We show the procedure to encode the LTL formula using TLA+ language. For each attack formula, the verification process generates a counterexample proving the occurrence of the formalized attack. These counterexamples model the execution sequence leading to the breach while providing key metrics such as the number of states generated, the number of pending states, the elapsed time, and the identification of redundant states. Based on the execution traces obtained, we formulate proposals for enhancing the specification of the MQTT protocol.


1 INTRODUCTION


The SIGIRO project aims to set up an intelligent system for the integrated management of water resources in the Marrakech-Safi and Tunisia's northwest regions in response to the challenges imposed by climate changes. The initiative is based on the systematic collection of data relating to the quality, quantity, and availability of water from various sources, including rivers, lakes, water tables, reservoirs and groundwater. This information will serve as a basis for planning climate change adaptation measures in the region. The data is collected via an IoT platform relying on IoT sensors. Monitoring the water management platform encompasses monitoring the IoT sen-

sors used, network, data exchanged, and communication protocols associated with water resource management. It ensures the correct functioning of the platform, including hardware and software components, data quality, security, and integrity. Indeed, compromised or incorrect data can have a significant impact on the quality of decisions taken. However, problems arise when protocols cannot preserve data integrity, when hardware is misconfigured, exposing it to the risk of intrusion, and when gaps are found in the management platform. In such circumstances, there is no guarantee, for example, that water level readings or pollution data will remain unchanged. The consequences of such problems can be severe, such as failure to detect pollution in water used for irrigation or incorrect programming of dam gate openings. That's why the SIGIRO project is deploying a systematic process to strengthen monitoring at the water management platform level. It aims

^a <https://orcid.org/0000-0002-1824-1238>

^b <https://orcid.org/0000-0001-6693-6352>

^c <https://orcid.org/0000-0002-0971-2368>

^d <https://orcid.org/0000-0002-0302-234X>

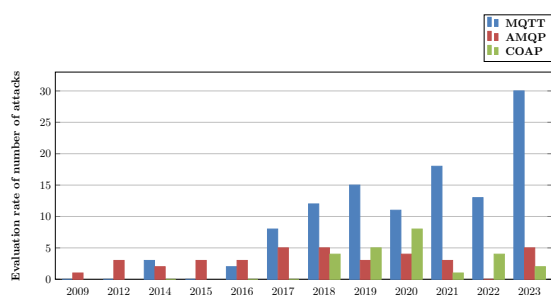


Figure 1: Evaluation rate of number of attacks by CVE.

to identify vulnerabilities and potential risks linked to climatic factors.

The rapid expansion of the Internet of Things (IoT) and the diversity of connected devices, often need more adequate security measures. According to the latest IoT analytic report published in May 2023 (IoT-Analytics, 2023), over 16.7 billion IoT devices are expected to be connected worldwide, heightening concern about the increase in attacks in this area. The NVD (National Vulnerability Database) is crucial in addressing this issue. This public database is managed by the National Institute of Standards and Technology (NIST) in the USA. The NVD collects information on software vulnerabilities from various sources, including security researchers, software manufacturers, government organizations, and public reports (NVD, 2023). The NVD uses CVE (Common Vulnerabilities and Exposures) to standardize reference vulnerabilities in its database, enabling IT security professionals to track, report, and resolve these vulnerabilities consistently and standardized (CVE, 2023). According to the latest data, there have been 1,210 IoT-related attacks since 2015. This significant increase in attacks reflects growing security concerns in the IoT field. In our IoT platform, data is transmitted via the Internet using IoT communication protocols such as CoAP, AMQP, and MQTT. As figure 1 shows, there is an annual assessment of the number of attacks discovered. With the CoAP protocol, 24 attacks we recorded since 2018. On the other hand, with the AMQP protocol, the number of attacks has increased to 37 since 2009. Finally, we observed a significant increase with the MQTT protocol, totaling 112 attacks. In 2023, domain experts discovered 30 new attacks linked to the MQTT protocol. This trend underlines the importance of remaining vigilant and reinforcing the security of IoT communications, particularly with the MQTT protocol.

We noted a crucial aspect in these sources, namely that the descriptions of attacks are informal, providing only information on the nature of the incidents. This poses challenges for attack prevention and re-

sponse, as the lack of detailed descriptions can make it challenging to identify appropriate countermeasures to avoid or mitigate attacks. Effective vulnerability and incident management often require formal documentation, as informal descriptions can lead to a non-deterministic interpretation. Such formal documentation would detail attacks and vulnerabilities and propose specific actions to prevent or mitigate risks. In this context, we set out to systematically explain intentional attacks, using a structured process to provide a formal description of a set of vulnerabilities.

Our description approach uses a temporal logic formalism called Linear Temporal Logic (LTL). In this article, we present a formalization of seven common attacks against the MQTT protocol, which led to the creation of 12 LTL formulas, each precisely representing a specific attack scenario. We then verified these formulas using the TLC simulator. TLC is a software tool widely used for model checking, particularly in computer systems and protocols. When a formula is violated, TLC generates a counterexample that models the execution sequence leading to the violation while providing four key metrics: the number of states generated, the number of distinct states found, the number of states waiting, and the elapsed time. Based on the execution traces obtained with TLC, it becomes possible to design a set of patches aimed at improving the protocol, which has a significant impact on IoT platform monitoring.

The remainder of this document is organized as follows: Section 2 provides an overview of previous work, reviewing related research and studies in the field. The following section outlines the essential preliminary elements for formalizing and verifying attack scenarios. It establishes a contextual basis by detailing two specific languages: the temporal logic formalization LTL and the algorithmic modeling language PlusCal-2. Section 4 examines the seven attack scenarios and their classifications and explains the associated formalization method. Section 5 discusses the formula verification process, illustrated by an example run. Finally, Section 6 includes an evaluation of the results obtained, concluding with a proposal for a set of corrections to be made to the specification.

2 RELATED WORK AND MOTIVATION

There are two main ways of verifying protocols: testing and formalizing properties.

Testing involves executing concrete tests to evaluate the protocol's behavior in various situations. The study in (Stijn et al., 2017) offers a detailed exam-

ple of applying this method. Researchers are working to develop and apply a formal approach to evaluating protocol implementations, also known as conformance testing. Specifically, they examine whether each protocol implementation conforms to standards moderated by regulations previously defined by the security community. In this article, the research team reviewed the implementation of the MQTT protocol, recently standardized by ISO. To carry out these compliance tests, the researchers used TTCN-3, a specification and execution language that has demonstrated its effectiveness in conducting compliance tests. The researchers evaluated the compliance of three open-source implementations of the MQTT protocol, namely Mosquitto, Emqtt, and RabbitMQ. The tests during this research involve sending messages to the SUT (System Under Test) and analyzing the corresponding responses. The researchers succeeded in automatically running 13 tests, each representing a test of a specific standard. According to these tests' results, all three evaluated implementations demonstrated partial compliance with the standard norms. While TTCN-3 can be crucial in validating telecommunications systems, it is not explicitly oriented toward modeling security aspects such as attacks or vulnerabilities.

Formalizing properties means expressing a system's characteristics or expected behaviors precisely and formally. Using proper analysis tools, such as model checking (UPPAL SMC, TLC), it is possible to check whether the system respects the specified formal properties automatically. In (Houimli et al., 2017), the authors provided a semi-formal modeling of the MQTT protocol using UML, followed by implementing a formal model with timed and probabilistic automata using the UPPAAL toolset. UPPAAL offers the possibility of verifying the MQTT protocol using its SMC model-checker for in-depth qualitative and quantitative analysis. However, a notable limitation of this approach lies in the general nature of the properties verified, which need to be specifically formulated to capture particular attack scenarios. While important, general properties such as permanence, security, reachability, number of active and inactive nodes, and message transfer and reception success rates are not necessarily designed to detect specific vulnerabilities to potential attacks. By focusing exclusively on general properties, there is a risk that particular attack scenarios will go undetected, as generic properties are not always suitable for identifying specific malicious behavior.

Temporal Logic of Actions (TLA+) is frequently used to specify distributed and concurrent systems. In (Akhtar and Zahoor, 2021), Akhtar and Zahoor ap-

plied a formal method based on TLA+ to verify the correctness of the MQTT protocol. They used the PlusCal-2 algorithmic language to define the three components of the protocol, which were then translated into formal TLA+ specifications. These specifications were submitted to the TLC model checker for validation of the safety and durability properties defined for the system. Precisely, the TLC model-checking tool reproduces specific unexpected system behaviors. Compared with existing methods, TLC enables a particular behavior or execution trace to be generated each time a property is violated. The authors have taken an initial step in attack formalization. They have formalized a specific attack scenario in which an attacker monitors the network traffic of the MQTT broker. The attacker intercepts a CONNECT message sent to the Broker by any client and substitutes his own CONNECT message instead. In this context, the Broker authorizes the connection, as no security measures are in place to detect or prevent such an attack.

In this context, we have proposed a set of formulas specifying seven distinct attacks targeting MQTT protocol services and their variants using Linear Temporal Logic (LTL). Our LTL-based formalization relies on the combination of event sequences modeling the moments when vulnerabilities are injected into the protocol during the execution of the attack scenarios. Before formalizing the attacks, we tested whether or not these attacks existed in the MQTT protocol using the Mosquito tool (Mosquitto, 2018). We have posted the code for the attack scenarios in the following link : <https://github.com/aminajandoubi/formalization-of-attacks-MQTT>. This formalization is of crucial importance for describing vulnerability injections. Further, we tested these formulas using the MQTT specification proposed by the researchers in (Akhtar and Zahoor, 2021). This specification offers the possibility of tracing the violation of a formula, enabling us to design solutions to prevent the realization of these attack scenarios.

3 PRELIMINARIES

This section recapitulates the essential preliminary elements for formalizing and verifying attack scenarios. It establishes a contextual basis by detailing two specific languages: the temporal logic formalization LTL and the algorithmic modeling language PlusCal-2.

3.1 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) is a formal language used to express and reason about systems' temporal properties. It is a type of temporal logic that deals with ordering events in time. It specifies the expected behavior of a transition system (TS) by selecting the only possible future (Baier and Katoen, 2008).

Let $K = (S, AP, L, R)$ be a Kripke structure, where S is a set of states, AP is a set of atomic propositions, L is a mapping $L : S \rightarrow 2^{AP}$ and R is a total relation.

The following grammar defines LTL:

$$\Phi \models \text{true} \mid \text{false} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid \circ\Phi \mid \Phi_1 \cup \Phi_2 \mid \Phi_1 \Rightarrow \Phi_2 \mid \Phi_1 \Leftrightarrow \Phi_2$$

The semantics of a temporal formula is provided by the satisfaction relation:

$$\text{TS} \models : (\mathbf{K} \times \mathbf{S} \times \text{FORMULA}) \rightarrow \{\text{True}, \text{False}\}$$

LTL is given by the standard boolean logic enhanced with temporal operators:

- \diamond : “eventually” (eventually in the future)
- \square : “always” (now and forever in the future)
- \neg : “Not” (negation)

3.2 PlusCal-2 Language

The PlusCal-2 version, developed by Sabina Akhtar, extends and perfects the PlusCal algorithmic modeling language, initially designed by Leslie Lamport under the name +Cal. Once a PlusCal-2 model has been established, it can be compiled in TLA+ to enable model verification using tools such as The TLC (The TLA+ Model Checker) (Akhtar et al., 2011). To facilitate the verification of properties, PlusCal-2 offers a section within its code structure dedicated to the definition of properties. In this section, we use a **temporal** term marking the beginning of a temporal assertion to express properties. PlusCal-2 uses temporal logic, more specifically TLA+ logic, to formulate and verify temporal properties within concurrent and distributed systems, according to the following grammar:

• Propositional Logic

Propositional logic is the mathematics of the two Boolean values **TRUE**/ **FALSE**, and the five operators (disjunction (or), conjunction (and), negation (not), implication (implies), equivalence (is equivalent to)) are as follows :

$$\vee \mid \wedge \mid \sim \mid \Rightarrow \mid \Leftrightarrow$$

• Sets

Sets denoted (S) , represent groupings of elements. They can be defined and manipulated in TLA+ specifications to model states and transitions within a system. The most common operations on sets are:

$$\cap \mid \cup \mid \subseteq \mid \setminus$$

• Predicate Logic

Once sets are available, asserting that a formula is valid for all or some elements becomes natural. Predicate logic extends propositional logic by introducing the two quantifiers :

$$\forall A \mid \exists E$$

The formula $\forall x \in S : F$, asserts that formula F is true for every element x in the set S . In contrast, the formula $\exists x \in S : F$ asserts that formula F is valid for at least one element x in S .

• Temporal Operators

Temporal operators are used to express temporal properties about the behavior of a system. Three of the most commonly used temporal operators are :

- ❖ $\Box F$: F is always true
- ❖ $\langle \rangle F$: F is eventually true

3.3 Rules for Transforming

Table 1 shows the equivalences between the symbols we have used in LTL and the operators used in TLA+:

- **“Temporal”**: This symbol is specific to TLA+ and represents the notion of time in logic. In LTL, there is no equivalent symbol.
- **“Eventually”**: indicates that a proposition will be true at some point in the future.
- **“ ψ ” and “ ϕ ”**: are used to represent atomic propositions, where $I.\psi = TRUE$ or $I.\phi = TRUE$ means that the proposition is true.
- **“Implication”**: indicates that if a proposition is true, then another proposition will also be true.
- **“Negation”**: is used to negate a proposition.

Table 1: Rules for transforming.

LTL	TLA+
	temporal
\diamond	$\langle \rangle$
ψ	$I.\psi = TRUE$
ϕ	$I.\phi = TRUE$
\Rightarrow	\Rightarrow
\neg	\sim

4 ATTACKS SPECIFICATIONS

This section examines the seven attack scenarios and their classifications and explains the associated formalization method.

4.1 Attack Scenarios

As a first step, we formalized seven common attacks targeting the MQTT protocol. The attack scenarios are detailed in (Wang et al., 2021) and (Jia et al., 2020). Table 2 shows the main characteristics of each attack scenario. The first three columns detail each scenario's identifier, the Attack Name, and impact. First, an attacker intentionally publishes malicious messages on a specific topic to mislead the system and induce a subscription. The second attack involves the interception and recording of communication data, followed by re-transmission, which can lead to various consequences such as identity theft, data manipulation, or violation of the integrity of the information exchanged. The malware can return nine different types of messages the MQTT client sends: CONNECT, PUBLISH, PUBREL, SUBSCRIBE, UNSUBSCRIBE, PUBACK, PUBREC, PUBCOMP, and DISCONNECT. The third attack involves identity theft. In the fourth attack, an attacker maliciously subscribes to topics, violating confidential data. The fifth attack consists of manipulating or surpassing a customer's will by injecting an unauthorized malicious connection message (will) into the topic dedicated to the will. The sixth attack involves maliciously injecting PUBLISH (retain) messages into an unauthorized issue. The last attack allows an attacker to take control of a client's session to access confidential data.

In the fourth column, we have specified the security attributes impacted by each attack. These attributes are essential to ensure confidentiality, integrity, availability, and other security-related aspects, where each attack can impact two aspects, either data or system behavior. As the table shows, attacks 1, 2 and 4 affect behavioral integrity and data confidentiality, while attacks 3 and 7 affect confidentiality. The last two attacks 5 and 6 only affect behavioral integrity.

Each attack scenario is characterized by a specific set of events. This is crucial, as it facilitates understanding of the sequential stages of the attack. More precisely, by examining these events, we can understand the different stages leading to the satisfaction of the attack scenario. This detailed understanding of the events associated with each attack scenario can contribute significantly to formalizing these attacks. In short, formalizing attacks based on this in-depth un-

derstanding of events offers a powerful means. It enables attacks to be simulated using a model checker and provides a formal, systematic system analysis.

4.2 Classification of Attack Scenarios

Table 3 presents a classification of the seven attack scenarios into four classes based on the LTL formula.

In our formalization approach, the formulas are divided into two types of properties, namely ϕ and ψ . The ϕ properties encompass events related to fault injection, while the ψ properties designate observational events concerning system behavior in the event of possible fault injections. The table shows that the classes are classified according to their implementation complexity. For example, class 1 is considered the simplest to implement, as it contains only injection properties. However, class 4 is the most complex, incorporating two observation properties, making it more delicate to implement. We can conclude that the presence of observation properties makes scenarios more challenging to realize.

We have created graphical representations for all the LTL formulas we have formulated.

4.2.1 Class C1

As depicted in Figure 2, the formula asserts that, at some future point, ϕ_1 will inevitably lead to ϕ_2 being true.



Figure 2: Graphic representation of C1.

4.2.2 Class C2

As we mentioned in figure 3, the formula is eventually true that if ψ is true, then ϕ will be accurate at some point in the future.

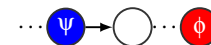


Figure 3: Graphic representation of C2.

4.2.3 Class C3

This formula ensures that at some time in the future, if ϕ_1 and ψ_2 become true, then ϕ_2 will necessarily be true later. Figure 4 shows the graphical representation of this formula. There are two possible representations due to the flexibility in the order of occurrence of the first two properties ϕ_1 and ψ .

4.2.4 Class C4

This formula means that if ψ_1 is accurate at a given time, then at a later time, ψ_2 will become true. If

Table 2: Characteristics of attack scenarios.

ID	Attack Name	Attack Effect	Security Attributes
1	Malicious Response Topic Publish	Force the subscription	<ul style="list-style-type: none"> Behavioral integrity Data confidentiality
2	Replay Attack	Violation of the integrity of exchanged information	<ul style="list-style-type: none"> Behavioral integrity Data confidentiality
3	Client Identity Hijacking	Usurpation of identity	<ul style="list-style-type: none"> Confidentiality
4	Malicious Topic Subscription	Duplication of subscription	<ul style="list-style-type: none"> Behavioral integrity Data confidentiality
5	Unauthorized Will Message	Publication via the connection	<ul style="list-style-type: none"> Behavioral integrity
6	Unauthorized Retained Message	Force publication (retain)	<ul style="list-style-type: none"> Behavioral integrity
7	Faults in Managing MQTT Sessions	Access to confidential data	<ul style="list-style-type: none"> Confidentiality

Table 3: Classification of attack scenarios.

	LTL formula	ID	Events		
			REP	ψ	ϕ
C1	$TS \models \diamond(\phi1 \Rightarrow \diamond\phi2)$	5			$\phi1 : \text{CONNECT}$
		6	⚡		$\phi2 : \text{PUBLISH}$
C2	$TS \models \diamond(\psi \Rightarrow \diamond\phi)$	2	⚡	msg	msg
		3		CONNECT	CONNECT
C3	$TS \models \diamond((\phi1 \wedge \diamond\psi) \Rightarrow \diamond\phi2)$	1		SUBSCRIBE	$\phi1 : \text{CONNECT}$
		4	⚡	SUBSCRIBE	$\phi1 : \text{CONNECT}$ $\phi2 : \text{SUBSCRIBE}$
C4	$TS \models \diamond((\psi1 \Rightarrow \diamond\psi2) \Rightarrow \diamond\phi)$	7	⚡	$\psi1 : \text{CONNECT}$ $\psi2 : \text{DISCONNECT}$	CONNECT

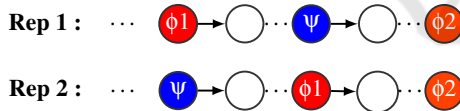


Figure 4: Graphic representation of C3.

this implication occurs, then at an even last time, ϕ will become true. A graphical representation of the formula is given in figure 5.



Figure 5: Graphic representation of C4.

5 ATTACKS VERIFICATION

This section outlines the steps in compiling the MQTT specification. We then put these concepts into practice by implementing an example. This example illustrates the compilation process by examining the attack scenario with ID 3 (Client identity hijacking).

5.1 Compilation Phases for the MQTT Specification

Compiling the MQTT specification takes place in three stages, as shown in the figure 6. The first phase involves drafting the various MQTT functionalities, formulating the LTL formula to be verified, and representing the attack scenario in the PlusCal-2 language. This step serves as a prerequisite for the next phase of the process. This next phase includes a PlusCal-2 compiler that translates these into TLA+ specifications. It is responsible for creating the TLA+ specification. More specifically, the primary role of this phase is to produce two files for the TLC model checker. The first file, called "TLA", contains the TLA+ specifications for the algorithm, while the second file, the configuration file, contains the algorithm's configuration parameters. We used the MQTT protocol specification written by Sabina (Akhtar and Zahoor, 2021). Sabina has modeled the protocol's roles: Broker, Publisher, and Subscriber. Our arti-

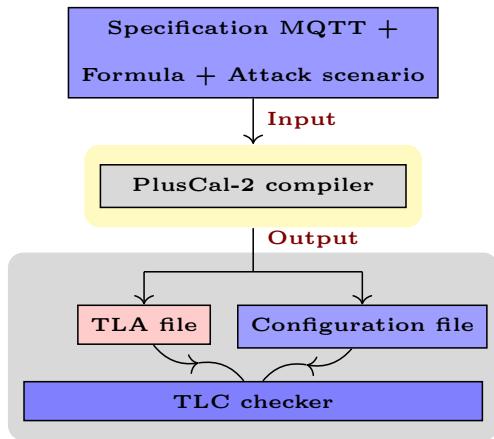


Figure 6: The compilation phases for the MQTT specification.

cle only includes the code for one attack scenario. We have grouped all the code for the other attack scenarios in the following link: <https://github.com/aminajandoubi/formalization-of-attacks-MQTT>.

5.2 Example of Verification of Scenario

We have chosen to illustrate this compilation process by examining one attack scenarios. In this attack an attacker can use the victim’s identity to connect to the server and take the victim offline. Figure 7 shows the structure of the input file. The first part of the file presents the three-component MQTT specification developed by Sabina (Akhtar and Zahoor, 2021). The second part covers the specification that models the attack scenario. The attack process begins by monitoring packets on the broker’s network. When the length of the list of "network[BrokerID]" packets is more significant than zero, it retrieves the first packet in the list, checks its type, records the sender’s identifier in the "pk" variable, then sends a new "CONNECT" packet to the "BrokerID" address. This attack can allow the attacker to access private information or disrupt communication between the client and the broker, leading to significant security and confidentiality risks in networks using MQTT. The final section of the file contains the formula to be checked, which in our example corresponds to the formula for class C2 explained in the table 3. In accordance with the transformation rules for the table 1, this LTL formula is presented in 1.

$$\text{temporal } \neg(\forall I \in \text{attaque} : (\mathbf{I.OBS} = \mathbf{TRUE} \Rightarrow \diamond(\mathbf{I.INJ} = \mathbf{TRUE}))) \quad (1)$$

We performed the negation of all formulas to identify possible counterexamples using the TLC checker. The result is shown in figure 8, clearly indicating

```

1  \\ Spec MQTT
2  \\ Spec attacker
3  process attaque[1]
4  CONNECT:
5    when Len(network[BrokerID]) > 0;
6    with packet = Head(network[BrokerID])
7    if packet.type = "CONNECT" then
8      pk := packet.sender;
9      network := send(BrokerID, [type |-> "CONNECT",
10     sender |-> packet.sender]);
11     network[BrokerID] := Tail(network[BrokerID]);
12     OBS := TRUE;
13     end if;
14   end with;
15 if OBS = TRUE then
16 INJ := TRUE;
17 break;
18 end if;
19 end loop;
20 end process
21 \\ Formula to be verified
  
```

Figure 7: Structure of the input file for the Client identity hijacking attack scenario.

the detection of a violation of the temporal properties specified in our input file. The "Error" section

```

1  TLC Model-checking
2  Error: Temporal properties were violated.
3  The following counter-example:
4  STATE 1: <Initial predicate>
5  .....
6  STATE 17: Back to state 16.
7  1297 states generated, 534 distinct states found, 0
8  states left on queue.Time elapsed:3.481(s)
  
```

Figure 8: Execution results produced by TLC model checker.

suggests that specific temporal properties were not respected during the state space exploration. Subsequently, the output generates a "counterexample" detailing 17 system states violating the specified properties. Within this sequence of conditions, we can observe the temporal evolution of a series of events. Initially, the system is in a state where no client is active, three subjects are in the pool, and an "attacker" process is defined, listening to the broker’s network. The following states allow processing to be applied until a state is found that helps the value of the OBS variable, which represents the ψ property, change to TRUE. In the following states, up to state 17, the INJ variable representing the ϕ property is changed to TRUE, which causes the TLC model checker to stop the state exploration process. The complete execution can be found at the following link: <https://github.com/aminajandoubi/formalization-of-attacks-MQTT>.

6 EVALUATION

This section includes an evaluation of the results obtained, concluding with a proposal for a set of corrections to be made to the specification.

6.1 Results and Discussion

We have compiled a table containing the information that appears to result from running the TLC model checker on the attack scenarios.

The three key measures in the table 4 are:

Table 4: Test results for attack scenarios.

			States generated	Distinct states found	Time elapsed (s)
C2	2	CONNECT	3606	1386	3.882
	2	PUBLISH	5046	1799	4.119
	2	PUBREL	593	235	2.496
	2	SUBSCRIBE	1081	431	2.636
	3		1297	534	3.481
C3	4		1800	669	4.234

- 1. Number of States Generated.** This first measure indicates the number of system states generated during model execution. This measure is essential for assessing the performance of model checking and indicates the time needed to complete the process.
- 2. Number of Distinct States Found.** The second measure represents the number of unique and different states among those generated. A high number of distinct states concerning the total number of states generated may suggest the presence of redundancies in the model, requiring optimizations to speed up the verification process.
- 3. Time Elapsed.** The fourth measure indicates the time that has elapsed since the start of the operation or process.

A fourth metric is the number of states remaining in the model checker's queue. As the tool explores system states, it places them in a queue for subsequent verification. In our verification of these attack scenarios, we obtained a value of zero for this metric, testifying to the effectiveness of the proof.

The results in the table show that we have successfully tested the attack scenarios with the respective identifiers 2, 3, and 4, except for five message types in attack 2: UNSUBSCRIBE, PUBACK, PUBREC, PUBCOMP, and DISCONNECT. In addition, we could not test the attacks on identifiers 1, 5, 6, and 7 due to the lack of certain features in the MQTT

protocol specification. This highlights the importance of updating the existing specification to include these missing aspects, allowing the verification model to consider these essential elements. By ensuring that the specification is updated correctly, we can close the identified gaps and guarantee more comprehensive protection against all classes of potential attacks.

6.2 Recommended Modifications

In this section, we have suggested modifications to the specification developed by researcher Sabina (Akhtar and Zahoor, 2021). All corrections apply to the code specifying the Broker process, which constantly listens on the network. As a first step, we've recommended adding a condition when a "CONNECT" packet is received, enabling verification of the client ID. This is intended to prevent class C2 and C3 attacks. The condition consists of checking the ID of each connection request received in the client list before accepting it. This measure aims to prevent the duplication of users with the same ID number, thus preventing the attacker from reusing messages in the event of a connection failure. In other words, adding this condition would reinforce system security by preventing both attacks based on reusing client ID numbers and repeated attempts to send the same message. Secondly, in line with scenario 4 of class C3, it would be necessary to incorporate a new index at the MQTT broker level listing all the clients in the network with which it needs to communicate. Before processing each connection request, checking whether the requester belongs to our network would be imperative. This verification would prevent attacks seeking to connect to the network, which could result in access to shared sensitive data or data tampering.

7 CONCLUSION

The observation that attack descriptions are often informal is of crucial importance in the security field. The lack of formality in these descriptions defines a significant challenge by restricting the clarity and precision necessary for a thorough understanding of incidents. This shortcoming compromises effective prevention and response measures, as it hinders the accurate identification of vulnerabilities and the formulation of appropriate countermeasures. This underlines the imperative need to formalize attack descriptions.

In this context, we have developed a proposal to remedy this shortcoming. We have introduced a systematic and structured process specifically designed to formalize a set of vulnerabilities. This approach

aims to create a robust basis for a deeper understanding of attacks, facilitating the implementation of more precise and appropriate security measures. By formalizing these descriptions, we intend to enhance the resilience of systems in the face of threats, thus contributing to the overall improvement of the security posture. We used a temporal logic formalism known as Linear Temporal Logic (LTL) for our formalization. This article outlines our approach to formalizing seven common attacks against the MQTT protocol, creating 12 distinct LTL formulas. Each of these formulas precisely represents a specific attack scenario. We have classified these attacks into four categories based on their formulation, where each class is characterized by two distinct property types, ϕ and ψ . The number of properties in each formulation varies according to the events modeling the attack scenario, thus presenting an essential key to attack classification.

Next, we implemented a set of rules to transform LTL formulations into TLA+ for verification using the TLC simulator. When a formula is violated, the TLC simulator generates a counter-example that models the execution sequence leading to the violation. We successfully tested the attack scenarios with identifiers 2, 3, and 4, except for five message types in attack 2: UNSUBSCRIBE, PUBACK, PUBREC, PUBCOMP, and DISCONNECT. In addition, we could not test the attacks on identifiers 1, 5, 6, and 7 due to the absence of certain functionalities in the MQTT protocol specification. This highlights the importance of updating the existing specification to include these missing aspects, allowing the verification model to consider these essential elements. By ensuring that the specification is updated correctly, we can close the identified gaps and provide more comprehensive protection against all classes of potential attacks. By analyzing the execution traces obtained with TLC, we have developed a set of patches to improve the specification.

For future systematic work, it is important to explore several avenues of research to improve security verification and extend the study. Firstly, developing the specification used to cover all the missing attack scenarios is imperative. Secondly, it is essential to consolidate all the attack scenarios in a single specification code and compare the results obtained.

ACKNOWLEDGEMENTS

The Moroccan-Tunisian Research and Development project PR&D-19/23: SIGIRO funds the work introduced in this paper. This project aims to monitor wa-

ter reservoirs in water-stressed regions.

REFERENCES

- Akhtar, S., Merz, S., and Quinson, M. (2011). A High-Level Language for Modeling Algorithms and Their Properties. In Davies, J., Silva, L., and Simao, A., editors, *Formal Methods: Foundations and Applications*, volume 6527, pages 49–63. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.
- Akhtar, S. and Zahoor, E. (2021). Formal Specification and Verification of MQTT Protocol in PlusCal-2. *Wireless Pers Commun*, 119(2):1589–1606.
- Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. The MIT Press, Cambridge, Mass. OCLC: ocn171152628.
- CVE (2023). CVE. page <https://cve.mitre.org>.
- Houimli, M., Kahloul, L., and Benaoun, S. (2017). Formal specification, verification and evaluation of the MQTT protocol in the Internet of Things. In *2017 International Conference on Mathematics and Information Technology (ICMIT)*, pages 214–221, Adrar, Algeria. IEEE.
- IoT-Analytics (2023). Iot analytics 2023. pages <https://iot-analytics.com/number-connected-iot-devices/>.
- Jia, Y., Xing, L., Mao, Y., Zhao, D., Wang, X., Zhao, S., and Zhang, Y. (2020). Burglars' IoT Paradise: Understanding and Mitigating Security Risks of General Messaging Protocols on IoT Clouds. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 465–481, San Francisco, CA, USA. IEEE.
- Mosquitto (2018). Eclipse Mosquitto. page <https://mosquitto.org/>.
- NVD (2023). Nvd., page <https://nvd.nist.gov/>.
- Stijn, v. W., Chris, M., and KPMG, C. (2017). Formal verification of the implementation of the mqtt protocol in iot devices. Amsterdam: University of Amsterdam. Amsterdam: University of Amsterdam.
- Wang, Q., Ji, S., Tian, Y., Zhang, X., Zhao, B., Kan, Y., Lin, C., Deng, S., Liu, A. X., and Beyah, R. (2021). MPInspector: A Systematic and Automatic Approach for Evaluating the Security of IoT Messaging Protocols. *30th USENIX Security Symposium*.