MLCA: A Model-Learning-Checking Approach for IoT Systems

Sébastien Salva(✉) and Elliott Blot

LIMOS - UMR CNRS 6158, Clermont Auvergne University, Clermont-Ferrand, France
sebastien.salva@uca.fr, eblot@isima.fr

Abstract. The Internet of Things (IoT) is a broad concept comprising a wide ecosystem of interconnected services and devices connected to the Internet. The IoT concept holds fabulous promises, but security aspects tend to be significant barriers for the adoption of large-scale IoT deployments. This paper proposes an approach to assist companies or organisations in the security audit of IoT systems. This approach called Model Learning and Checking Approach (MLCA) combines model learning for automatically extracting models from event logs, and model checking for verifying whether security properties, given under the form of generic LTL formulas hold on models. The originality of MLCA lies in the fact that auditors do not have to craft models or to be expert LTL users. The LTL formula instantiation, which makes security properties concrete, is indeed semi-automatically performed by means of an expert system composed of inference rules. The latter encode some expert knowledge, which can be applied again to the same kind of systems with less efforts. We evaluated MLCA on 5 IoT systems with security measures provided by the European ENISA institute. We show that MLCA is very effective in detecting security issues and provides results within reasonable time.

Keywords: Model learning · Model checking · Expert system · IoT software systems

1 Introduction

Using the Internet of Things (IoT) to stimulate transformational efficiencies in several application domains among which manufacturing, automotive, health and smart cities, is an idea that holds fabulous promises. Indeed, exploiting smart and connected devices to produce real-time data and to quicker take decisions provides new ways to make businesses more efficient, and to forge links between the digital world and the real. On the other hand, after the myriad of cyberattacks on IoT systems revealed during the few past years, experts have warned that IoT could harm people if IoT is left unsecured. As many technological concepts are involved under the IoT umbrella, it is indeed not surprising to observe that IoT systems are vulnerable to a wide range of security attacks. And it is likely that this security problem will grow more complex in the future, as long as new technologies and platforms will be proposed.

Many companies or organisations have started to be aware about the importance of including cyber-security in their IoT solutions. Many of them assess the risks of their

M. van Sinderen et al. (Eds.): ICSOFT 2020, CCIS 1447, pp. 70–97, 2021.
https://doi.org/10.1007/978-3-030-83007-6_4

IoT-based services and platforms by means of security audits. There are many ways to carry out an audit depending on the scope and objectives of the audit itself and on the resources allocated by the company. But an audit process usually follows many manual steps such as Define the audit objective, Collect software usage data, or Test the systems and applications methodically. Most of these activities are time-consuming and sometimes challenging. Fortunately, some templates or documents can be used to guide auditors in these steps. For instance, the National Institute of Standards and Technology (NIST) has proposed a framework made up of several stages, which can define the audit plan [26]. The European Telecommunications Standards Institute (ETSI) has proposed a general method and activities dedicated to undertake testing and risk assessment activities for large scale, networked systems [12]. Others documents related to security measures, good practices, and threats taxonomy e.g., the reports provided by the ENISA, Cloud Security Alliance, or OWASP organisations [9,11,27] are proposed to help derive objectives. The standard ISO/IEC 27030 [17], due to be published in 2022, will provide guidelines for security and privacy in IoT systems. Finally, testing guides, e.g., [19,27], along with testing tools e.g., [8,25] may be used to conduct the review.

Despite the strong benefits brought by these approaches, the efforts required for understanding how an IoT system under audit (SUA) is structured and behaves or for generating security tests is yet tremendous. These observations motivated us to present in [29] an approach called MLCA combining model learning and model checking to assist auditors in the understanding of SUA by means of models, and in the verification of security properties on these models:

– *Model Learning:* our approach starts by recovering formal models from an event log. We use the Labelled Transition System (LTS) to express the behaviours of every component (devices, servers, etc.) of SUA. These models can be used as documentation or to comprehend the functioning of the components and their interactions;
– *Property Instantiation and Model Checking:* our approach also takes as inputs generic security properties, which can be used independently of SUA. Usually, such properties have to be adapted for every model so that they share the same alphabet. This activity is known to be difficult and time consuming. To make it easier, MLCA helps auditors make them concrete by means of an expert system composed of rules, which encode some expert knwoledge about IoT systems. Then, our approach checks whether the LTSs satisfy the security properties, and returns counterexamples when issues are detected. The counterexamples may be used to interpret the results and provide countermeasures.

Contributions: This paper presents an extension of the MLCA algorithms given in [29], which mainly aims at improving both effectiveness and performance. We indeed showed that MLCA requires a manual inspection of the generated models to detect inconsistencies. We propose to enhance MLCA with a new step for assisting auditors in this model inspection. Besides, we provide a new security property instantiation algorithm, which generates less concrete properties. Consequently, this algorithm allows to significantly save time during the model checking step. Furthermore, this paper provides more details about the generation of concrete security properties. This paper also provides an empirical evaluation, which investigates the sensitivity (ratio of true positives) and specificity (ratio of true negatives) of MLCA and its performance in terms of execution times. We also compare the MLCA of [29] with this new version. This empirical evaluation was carried out on event logs collected from 5 IoT systems. This evaluation shows that our approach allows the detection of security issues with few false positives or negatives within reasonable time delays.

Paper Organisation: The paper is organised as follows: Sect. 2 discusses related work and presents our motivations. Section 3 offers an overview of the functioning of MLCA with a real example of IoT system. The MLCA's algorithms are detailed in Sect. 4. We recall some basic definitions and describe the five steps of the approach. The next section examines experimental results and discusses about the threats to validity. Section 6 summarises our contributions and draws some perspectives for future work.

2 Related Work

2.1 IoT Audit

A plethora of surveys or papers have exposed the opportunities, challenges, requirements, threats or vulnerabilities involved in the IoT security. Among them, several approaches have been proposed to audit IoT systems. The security audit of IoT devices is carried out with check lists or threat models in [19,25]. These lists or models have been devised or extended from the recommendations published by the OWASP organisation [27]. Other works focused on the IoT device audit by decrypting the traffic sent via TLS [33] so that the TLS traffic can be verified without compromising future traffic. This king of technique could be used prior to MLCA to obtain readable event logs.

Many approaches also rely on models to analyse the security of IoT systems, because models offer the advantage of expressing systems without ambiguity. In [14], security models are devised with data collected from an IoT system. A manual security analysis is then performed to find potential attack scenarios, evaluate security metrics and assess the effectiveness of different defence strategies.

Other works introduced specialised model-based testing (MbT) methods. Some of them are said to be active, i.e. security test cases are built by hands or automatically generated from a given (formal) specification, and are later used to experiment IoT systems [1,15,22,23]. These active testing techniques could complement MLCA to get larger event logs. Other methods are said to be passive because they are based upon monitoring tools, which detect the violation of security properties by checking rule satisfiability in the long run [6,20,32].

Several papers addressed the detection of security issues in IoT systems by means of model checking. The tool IoTSAT [24] is a SMT based framework, which analyses the IoT system security. IoTSAT models the device-level interactions as in our approach, but also policy-level behaviours and network-level dependencies. SOTERIA [5] is another model checking tool for IoT software. State-models are extracted from source code, then SOTERIA checks whether concrete security properties hold on these models.

In comparison to our approach, the works [14,23] go further in the risk assessment by proposing the evaluation of metrics. IoTSAT also goes further in the modelling of the

IoT system environment. But all of these approaches require models or formal properties, which have to be manually devised. SOTERIA offers the advantage of recovering state-models on condition that the source code of every component is available. But, the concrete security properties have to be written by hands. In contrast, MLCA generates behavioural models and dependency graphs from event logs. Besides, our approach semi-automatically instantiates generic properties with an expert system. These generic properties can be reused with several IoT systems. We provide a list of generic properties derived from the security measures proposed by the ENISA institute.

2.2 Model Learning

MLCA uses a passive model learning algorithm to recover formal models from event logs. Some papers also presented model learning approaches specialised to communicating systems in the literature [3,21,30]. Mariani et al. proposed in [21] an automatic detection of failures in log files by means of model learning. The approach segments an event log with two strategies: per component or per user. The former can be used with communicating systems to generates one model for each component. CSight [3] is another tool specialised in the model learning of communicating systems. The main contribution proposed by CSight lies in the mining of invariants in logs to improve the model precision. Unfortunately, invariant mining limits the application of this algorithm to small trace sets only.

We recently proposed the model learning approach CkTail in [30] and presented a comparison of CkTail with the previous passive techniques. In summary, we showed that CkTail builds more precise models by means of its trace segmentation algorithm, which tries to recover user sessions. Besides, compared to CSight, CkTail requires less constraints. Furthermore, CkTail is the only approach that detects component dependencies and expresses them with dependency graphs. The latter are used in MLCA to instantiate security properties.

3 MLCA Overview

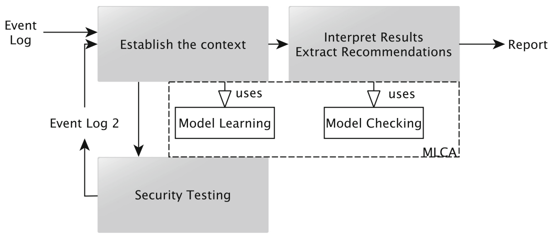

Fig. 1. Integration of MLCA with some audit stages (in grey) [29].

MLCA aims at assisting auditors to verify whether security properties hold on IoT systems with the automatic generation of behavioural models from event logs and with the generation of concrete security properties. Our approach can complement several existing security audit processes, e.g., the NIST or ETSI security audit frameworks. Figure 1 illustrates its integration. Most of the security audit processes include a step allowing auditors to understand the system context. We call it "Establish the context" in the figure. This step is often manually done by interpreting diverse documents, event logs, or by using scanning tools. With MLCA, a model learning algorithm is used instead to recover one behavioural model for every component of the IoT system from event logs. These models make the system understanding easier. They can also be given to MbT approaches for assessing the IoT system security with tests. While testing, more logs may be collected and hence new models can be re-generated to capture more behaviours.

In the meantime, these formal models can be analysed in an exhaustive manner to detect further security issues. This analysis is usually automatically performed by means of security properties modelled with formulas, which are evaluated with a model-checker. These formulas may express different security aspects, e.g., vulnerabilities. We focus in this paper on formulas expressing security measures used to protect an IoT system against threats. Several papers, e.g., [18,27,34] propose lists of recommendations dedicated to IoT systems, which can be used as security measures. We chose to consider the works proposed by the ENISA organisation as they gather the security measures suggested in several papers and works of other organisations.

3.1 The ENISA Security Measures

The ENISA organisation issued several documents exposing guidelines and security measures to implement secure IoT software systems with regard to different contexts (smart plants, hospitals, clouds, etc.). These measures correspond to high-level recommendations for developers, operators and security experts, which help improve the security level of IoT devices and communications among them.

The security measures provided by ENISA come from several other documents written by different organisations or institutes, e.g., ISO, IETF, NIST or Microsoft. We have chosen to focus on the paper related to security baselines in the context of critical information infrastructures [11]. This document gathers 57 security technical measures that should be *active* in an IoT system. As the models generated by our approach mostly express exchanges among components, we formulated the 12 security measures related to communications, which cover the following domains: Authentication, Privacy, Secure and Trusted Communications, Access Control, Secure Interfaces and Network Services, Secure Input and Output Handling, Trust and Integrity Management. These security measures are given in Table 1 (left side). We have formulated them with LTL formulas composed of predicates. Following the terminology used in [2], we call these formulas *property types*. The definitions and explanations of the LTL operators are given in Sect. 4. These property types are formulated by means of the predicates given in Table 2.

Table 1. Some ENISA security measures and LTL formulas expressing them.

GP-TM-18: the device software/firmware has the ability to update Over-The-Air (OTA), the update server is secure, the update file is transmitted via a secure connection, it does not contain sensitive data and is encrypted	$F\,getUpdate(f) \wedge (\mathrm{G}(getUpdate(f) \rightarrow (encrypted(f) \wedge \neg sensitive(f)))) \wedge \mathrm{G}((begin \wedge F\,end) \rightarrow (\neg(getUpdate(f) \wedge from(c))\,\mathrm{U}(authenticated(c) \vee end)))$
GP-TM-19: offer an automatic firmware update mechanism	$F\,searchUpdate \wedge (F\,searchUpdate \rightarrow \neg(F\,searchUpdate \rightarrow (\neg searchUpdate\,\mathrm{U}(input \wedge cmdSearchUpdate \wedge \neg searchUpdate))))$
GP-TM-22: ensure that weak, null or blank passwords are not allowed	$\mathrm{G}((Request \wedge from(c1) \wedge to(c2) \wedge Weakpass(x)) \rightarrow ((\neg(Response \wedge to(c1) \wedge from(c2)))\,\mathrm{U}((Response \wedge to(c1) \wedge from(c2)) \wedge (errorResponse \vee (\neg validResponse \wedge Response)))))$
GP-TM-24: credentials are encrypted during authentication	$\mathrm{G}((loginAttempt(c) \wedge credential(x)) \rightarrow encrypted(x))$
GP-TM-25: protect against abusive login attempts	$\mathrm{G}((begin \wedge F\,end) \rightarrow (\neg(\mathrm{G}((begin \wedge F(end \vee authenticated(c)) \rightarrow P(5,c))) \rightarrow (\neg end\,\mathrm{U}\,lockout(c)))\,\mathrm{U}\,end)$ with $P(n,c) = ((\neg(loginFail(c)) \wedge \neg(end \vee authenticated(c)))\,\mathrm{U}(end \vee authenticated(c) \vee ((loginFail(c) \wedge \neg(end \vee authenticated(c)))\,\mathrm{U}(end \vee authenticated(c) \vee P(n-1,c)))))$ for $n > 0$, and $P(0,c) = (\neg(loginFail(c))\,\mathrm{U}(end \vee authenticated(c)))$
GP-TM-26: password recovery system does not supply an attacker with information indicating a valid account	$\mathrm{G}(passwordRecovery \rightarrow \neg blackListedWord(x))$
GP-TM-34,38: ensure a proper and effective use of cryptography	$\mathrm{G}(sensitive(x) \rightarrow encrypted(x))$
GP-TM-42: do not trust data received and always verify any interconnections	$\mathrm{G}((\neg(validResponse \wedge to(c) \wedge \neg loginAttempt(c))\,\mathrm{U}(authenticated(c))) \wedge (\neg(Request \wedge to(c) \wedge \neg loginAttempt(c))\,\mathrm{U}(authenticated(c))))$
GP-TM-48: if a single device is compromised, it does not affect the whole set	$\neg\mathrm{G}(from(d) \wedge Unavailable) \rightarrow \neg(\neg output\,\mathrm{U}(output \wedge Unavailable)$
GP-TM-52(1): ensure a device is not susceptible to XSS	$\mathrm{G}((Request \wedge from(c1) \wedge to(c2) \wedge XSS(x)) \rightarrow \neg(\neg(Response \wedge to(c1) \wedge from(c2))\,\mathrm{U}(Response \wedge to(c1) \wedge from(c2)) \wedge (errorResponse \vee (\neg validResponse \wedge Response))))$
GP-TM-52(2): ensure a device is not susceptible to SQL injection	$\mathrm{G}((Request \wedge from(c1) \wedge to(c2) \wedge SQLinjection(x)) \rightarrow \neg(\neg(Response \wedge to(c1) \wedge from(c2))\,\mathrm{U}(Response \wedge to(c1) \wedge from(c2)) \wedge (errorResponse \vee (\neg validResponse \wedge Response))))$
GP-TM-53: avoid security issues when designing error messages	$\mathrm{G}((errorResponse \vee \neg validResponse) \rightarrow \neg blackListedWord(x))$

3.2 MLCA Requirements

The capability of MLCA in auditing IoT systems depends on several realistic assumptions made on a system under audit denoted SUA:

- **A1 Event Log:** we consider the components of SUA as black-boxes. We assume that each device, server, or gateway is physically secured and that we only have access to the network or user interfaces through the network. Event logs include timestamps given by a global clock. We consider having one event log;
- **A2 Message Content:** components produce events that include parameter assignments allowing to identify the source and the destination. Other parameter assignments may be used to encode data. Besides, an event is either identified as a request or a response. Many protocols, e.g. HTTP, allow to easily distinguish both of them;
- **A3 Device Collaboration:** to learn precise models, we want to recognise sessions of the system in event logs. We consider two exclusive cases:
  - **A31:** the components of SUA cannot run multiple instances; requests are processed by a component on a first-come, first served basis. Besides, the

Table 2. Predicates defined from 12 ENISA measures related to communications.

Predicate	Short Description
Begin	Beginning of a new session
End	End of a session
$From(c)$	Event coming from c
$To(c)$	Event sent to c
Request	Event is a request
Response	Event is a response
Input	Event is an input
Output	Event is an output
GetUpdate(x)	Response including an update file
CmdSearch-Update	The component received the order to search for an update
$Sensitive(x)$	Data x is sensitive
$Credential(x)$	Data x is acredential
$Encrypted(x)$	Data x is encrypted
SearchUpdate	Component requests for an update
$LoginAttempt(c)$	Authentication attempt with c
$Authenticated(c)$	Successful authentication with c
$LoginFail(c)$	Failed authentication with c
$Lockout(c)$	c is locked due to repetitive authentication failures
PasswordRecovery	Component uses a password recovery mechanism
$BlackListedWord(x)$	Message x includes black listed words
ValidResponse	Correct response with correct status
ErrorResponse	Response containing an error message
Unavailable	Component that received the request is unavailable
$XSS(x)$	Data x includes an XSS attack
$SQLinjection(x)$	Data x includes an SQL injection attack
$Weakpass(x)$	Password x is weak or blanck

components follow the request–response exchange pattern (a response is associated with one request, a request is associated with responses), or

- **A32:** the events that belong to the same session are identified by a unique parameter assignment.

The session recognition mentioned in A3 helps extract execution traces expressing complete behaviours of SUA, i.e. disjoint action sequences starting from one of its initial states and ending in one of its final states. A32 expresses that messages include an identifier allowing to observe whole collaborations among components. Usually, the session identification strongly facilitates the trace extraction. Unfortunately, we have observed that this technique is seldom adopted with IoT systems. Hence, when there is no session identifier, we restrict the functioning of SUA with A31. We have observed that this assumption can be applied with many wireless or IoT systems.

3.3 A Motivating Example

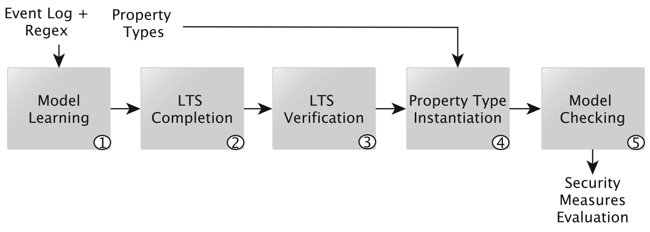

Fig. 2. The MLCA's steps.

We now present our Model-Learning-Checking approach MLCA, which aims at helping audit the security of SUA. It takes as inputs an event log, regular expressions allowing to format the event log, and generic security properties given under the form of property types. These property types have a pattern-level form, and have to be instantiated to make them concrete before being evaluated.

Figure 2 illustrates the 5 successive steps of MLCA. It starts by learning models from the event log. In short, the event log is firstly formatted by means of the regular expressions into a sequence of actions of the form $a(\alpha)$ with a a label and α some parameter assignments. In reference to A1, A2, an action $a(\alpha)$ indicates the sources and destinations of the messages with two parameters $from$ and to. The other parameter assignments capture data, e.g., acknowledgements or sensor data. Figure 3 illustrates a simple example of event log along with a regular expression allowing to format the events into actions. This expression retrieves a label (Req or Resp) and parameters in the messages (from, to and the remaining data, e.g., cmd:=auth). Figure 4 shows an example of action sequence where the first five actions are derived from the events of Fig. 3.

Then, we call the model learning algorithm CkTail to build one LTS $\mathscr{L}(c1)$ for every component $c1$ of SUA detected in the action sequence. $\mathscr{L}(c1)$ shows the behaviours of $c1$ in terms of messages exchanged with the other components. Besides, CkTail generates one dependency graph $Dg(c1)$ expressing how $c1$ interacts with the other components of SUA. The detailed functioning of our model learning algorithm is given in [30], but the auditor does not need to be aware of the details. Figures 5 and 6 illustrate the LTSs and dependency graphs obtained from our action sequence example. From these models, it becomes easier to understand that the system consists of four components: $c4$ is an application sending commands (from users), $c1$ and $c2$ are gateways and c_3 is an actuator (a smart light bulb). We can deduce from $\mathscr{L}(c1)$ and $\mathscr{L}(c2)$ that the first gateway authenticates to the second one after the receipt of commands from $c4$. Then, $c1$ sends to c_2 the state of a motion sensor, which seems to be integrated

```
Jun 08, 2019 14:16:21.521 CET;from:=c4;to:=c1 Req?cmd:=auth
Jun 08, 2019 14:16:21.758 CET;from:=c1;to:=c2 Req?login:=toto&password:=1234
Jun 08, 2019 14:16:22.136 CET;from:=c2;to:=c1 Resp response:=OK
Jun 08, 2019 14:16:22.385 CET;from:=c1;to:=c4 Resp response:=OK
Jun 08, 2019 14:16:23.287 CET;from:=c1;to:=c2 Req?presence:=1&switch:=On

Example of regular expression:
^(?<date>\w{3} \d{2}, \d{4} \d{2}:\d{2}:\d{2}.\d{3})\s(?<param1>[^;]+);
(?<param2>[\s]+)\s(?<label>[^?]+)?(?<param3>[^&]+)$
```

Fig. 3. Example of 5 HTTP messages collected from an IoT system. The regular expression retrieves a label and 3 parameters here. The label expression will be the label of the action in the action sequence *S*.

```
Req(from:=c4,to:=c1,cmd:=auth)
Req(from:=c1,to:=c2,login:=toto,password:=1234)
Resp(from:=c2,to:=c1,response:=OK)
Resp(from:=c1,to:=c4,response:=OK)
Req(from:=c1,to:=c2,presence:=1,switch:=On)
Req(from:=c4,to:=c1,cmd:=Light3min)
Resp(from:=c2,to:=c1,response:=OK)
Req(from:=c1,to:=c3,switch:=On)
Resp(from:=c3,to:=c1,response:=OK)
Req(from:=c2,to:=c3,switch:=On)
Resp(from:=c1,to:=c4,response:=OK)
Resp(from:=c3,to:=c2,response:=OK)
Req(from:=c1,to:=c3,switch:=Off)
Resp(from:=c3,to:=c1,response:=OK)
```

Fig. 4. Action sequence S.

to the gateway $c1$. According to the motion sensor state, $c2$ finally sends the command "switch:=on" to the smart light bulb $c3$.

With these models, it becomes easier to interpret the behaviours of SUA. Furthermore, as we now have formal models, different kinds of activities may be automatically or semi-automatically conducted to discover defects. In particular, the remaining steps of MLCA aims at checking whether property types hold on those LTSs. But, the LTSs and property types do not yet share the same alphabet, as the property types are generic. The auditor should re-formulate them for every LTS with the actions labelled on transitions. Instead, our approach automatically instantiates property types with the three next steps. More, precisely, given an LTS $\mathscr{L}(c1)$, the step "LTS Completion" automatically extends the LTS semantics; it analyses the LTS paths and injects new labels on transitions. These labels correspond to some predicates of the property types whose variables are assigned to concrete values. The step automates the label injection by using an expert system made up of inference rules, which encode some expert knowledge about SUA. The step produces a new LTS $\mathscr{L}'(c1)$. If we take back our example, this step completes the LTS $\mathscr{L}(c2)$ of Fig. 5 and gives the new LTS $\mathscr{L}'(c2)$ illustrated in Fig. 7. The transitions of $\mathscr{L}'(c2)$ are still labelled by the actions of the original LTS, but several new labels appeared. For instance, the expert system has detected a login attempt to $c1$ with the credentials $\{login := toto, password := 1234\}$, which are also recognised as sensitive data.

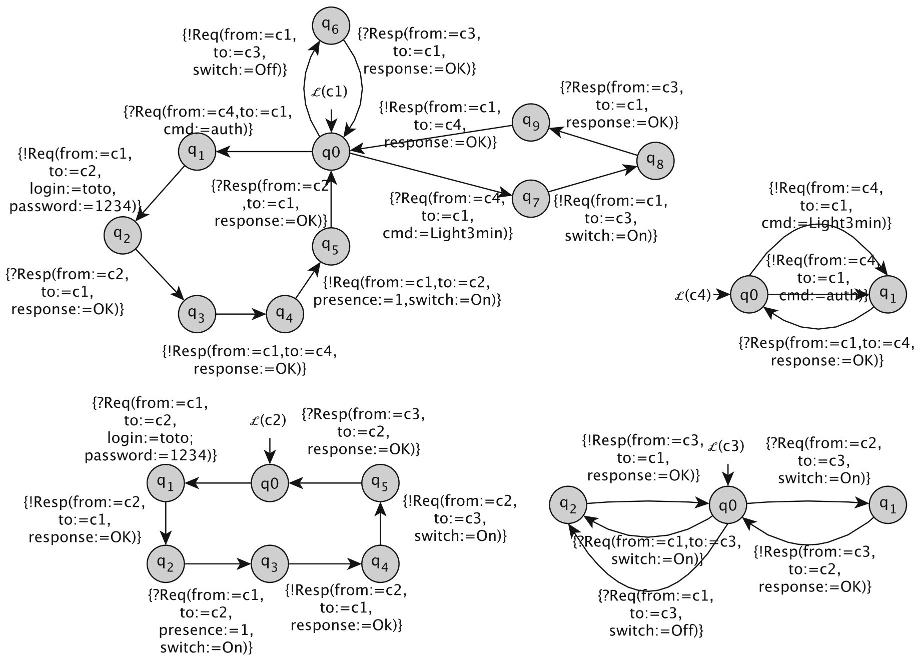
{!Req(from:=c1, to:=c3, switch:=Off)}
{?Resp(from:=c3, to:=c1, response:=OK)}
ℒ(c1)
{?Req(from:=c4,to:=c1, cmd:=auth)}
{!Resp(from:=c1, to:=c4, response:=OK)}
{?Resp(from:=c3, to:=c1, response:=OK)}
{!Req(from:=c1, to:=c2, login:=toto, password:=1234)}
{?Resp(from:=c2 ,to:=c1, response:=OK)}
{?Req(from:=c4, to:=c1, cmd:=Light3min)}
{!Req(from:=c1, to:=c3, switch:=On)}
{?Resp(from:=c2, to:=c1, response:=OK)}
{!Req(from:=c1,to:=c2, presence:=1,switch:=On)}
{!Resp(from:=c1,to:=c4, response:=OK)}
{!Req(from:=c4, to:=c1, cmd:=Light3min)}
{!Req(from:=c4, to:=c1, cmd:=auth)}
ℒ(c4)
{?Resp(from:=c1,to:=c4, response:=OK)}
{?Req(from:=c1, to:=c2, login:=toto; password:=1234)}
ℒ(c2)
{?Resp(from:=c3, to:=c2, response:=OK)}
{!Resp(from:=c2, to:=c1, response:=OK)}
{!Req(from:=c2, to:=c3, switch:=On)}
{?Req(from:=c1, to:=c2, presence:=1, switch:=On)}
{!Resp(from:=c2, to:=c1, response:=Ok)}
{!Resp(from:=c3, to:=c1, response:=OK)}
ℒ(c3)
{?Req(from:=c2, to:=c3, switch:=On)}
{?Req(from:=c1,to:=c3, switch:=On)}
{!Resp(from:=c3, to:=c2, response:=OK)}
{?Req(from:=c1, to:=c3, switch:=Off)}
q0
q1
q2
q3
q4
q5
q6
q7
q8
q9

Fig. 5. LTSs generated from the action sequence of Fig. 4.

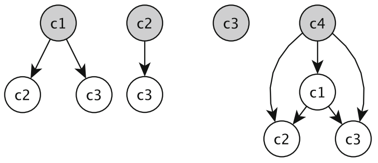

Fig. 6. Dependency graphs generated from the action sequence of Fig. 4.

The step “LTS Verification” helps auditors check the correctness of the previous step. Indeed, the expert system used to enrich the LTS transitions may suffer from the classical data acquisition problem. In other words, it may have inaccurate rules or missing ones. To help auditors check the LTS completion correctness, this step returns the list of predicates that have not been added on LTS transitions. Besides, it check the satisfiability of invariants on LTSs. These invariants allow the detection of some incorrect LTS transition completions (e.g., a transition whose action is neither a request nor a response) or missing labels (e.g., every transition must carry either an input or an output). These transitions are also retuned to auditors.

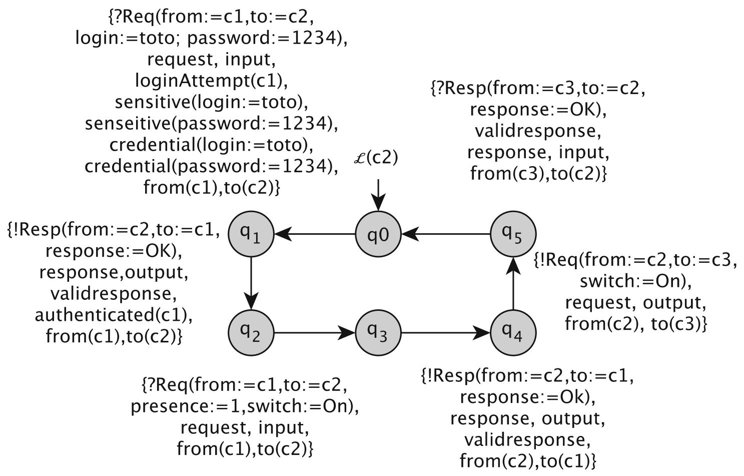
{?Req(from:=c1,to:=c2, login:=toto; password:=1234), request, input, loginAttempt(c1), sensitive(login:=toto), senseitive(password:=1234), credential(login:=toto), credential(password:=1234), from(c1),to(c2)}
ℒ(c2)
{?Resp(from:=c3,to:=c2, response:=OK), validresponse, response, input, from(c3),to(c2)}
q1
q0
q5
{!Resp(from:=c2,to:=c1, response:=OK), response,output, validresponse, authenticated(c1), from(c1),to(c2)}
{!Req(from:=c2,to:=c3, switch:=On), request, output, from(c2), to(c3)}
q2
q3
q4
{?Req(from:=c1,to:=c2, presence:=1,switch:=On), request, input, from(c1),to(c2)}
{!Resp(from:=c2,to:=c1, response:=Ok), response, output, validresponse, from(c2),to(c1)}

Fig. 7. Example of LTS completion. New propositions (Begin, End, Credential, Sensitive, ValidResponse, etc.) are injected on transitions.

The next step "Property type instantiation" covers every new LTS and automatically instantiates the property types. This step returns a set of LTL formulas $\mathscr{P}(\mathscr{L}'(c1))$ exclusively written with atomic propositions. We call them *property instances*. These correspond to concrete security properties. Let's illustrate this step with the LTS $\mathscr{L}'(c2)$ of Fig. 7 and the property type $\mathbf{G}((loginAttempt(c) \wedge credential(x)) \rightarrow encrypted(x))$ derived from the ENISA measure GP-TM-24. By covering the labels of $\mathscr{L}'(c2)$, we obtain $Dom(c) = \{c1\}$ and $Dom(x) = \{login := toto, password := 1234\}$. Two property instances are then derived.

The final step is more classic as it calls a model-checker to verify whether an LTS $\mathscr{L}'(c1)$ satisfies the LTL formula of $\mathscr{P}(\mathscr{L}'(c1))$. The model-checker either returns true if a property instance holds or a counterexample path that violates it. This counterexample is particularly useful to understand why a component $c1$ does not meet a security property and should help localise a problem in the LTS $\mathscr{L}'(c1)$. Figure 8 shows a example of results returned by the model-checker NuSMV [7] after having evaluated if the LTS $\mathscr{L}'(c2)$ satisfies some property instances derived from five security measures. A property instance related to GP-TM-24 does not hold because both the login and password are not encrypted. The interpretation of the counterexample helps deduce that the credentials are not encrypted (Encrypted(login:=toto), Encrypted(password:=1234) not found). Such counterexamples may be used to develop an audit report, which should include recommendations or treatments to security issues.

After this overview of MLCA, we now develop its theoretical background along with its algorithms in the next section. It is worth noting that a user does not need to be aware of these details. He/she only needs to have an event log, a list of regular expressions and our list of property types.

```
...                                             ...

Property GP-TM-24(c1,login:=toto) is false:     Property GP-TM-38(login:=toto) is false:
counterexample:                                 counterexample:
-q0 -> q1                                       -q0 -> q1
  !Req(from:=c1,to:=c2,login:=toto;               !Req(from:=c1,to:=c2,login:=toto;
password:=1234),                                password:=1234),
  Request,                                        Request,
  output,                                         output,
  loginAttempt(c2),                               loginAttempt(c2),
  begin,                                          begin,
  sensitive(login:=toto),                         sensitive(login:=toto),
  sensitive(password:=1234),                      sensitive(password:=1234),
  credential(login:=toto),                        credential(login:=toto),
  credential(password:=1234)                      credential(password:=1234)
  from(c1),                                       from(c1),
  to(c2)                                          to(c2)
-q1 -> q2                                       -q1 -> q2
...                                             ...

Property GP-TM-42(c3) is false:
counterexample:                                 Property GP-TM-42(c1) is true
-q0 -> q1
...                                             Property GP-TM-53 is true
-q4 -> q5
  !Req(from:=c2,to:=c3,switch:=On),             ...
  request,
  output,
  from(c2),
  to(c3)
```

Fig. 8. Example of results obtained with the verification of five ENISA measures on the LTS $\mathscr{L}(c2)$. The counterexamples shows the LTS paths that do not satisfy the property instances.

4 The Model Learning Checking Approach

This section details the MLCA's steps 2 to 5. More details of the model learning step are available in [29, 30]. Before describing these steps, we provide some definitions and notations to be used throughout the remainder of the paper.

4.1 Preliminary Definitions

As in many works dealing with the modelling of atomic components, e.g., [4, 13], we express the behaviours of components with the well established Labelled Transition System (LTS) model. An LTS is defined in terms of states and transitions labelled by actions, themselves taken from a general action set $\mathcal{L}$.

Definition 1 (LTS). *A Labelled Transition System (LTS) is a 4-tuple $\langle Q, q0, \Sigma, \rightarrow \rangle$ where:*

- *Q is a finite set of states;*
- *$q0$ is the initial state;*
- *$\Sigma \subseteq \mathcal{L}$ is a finite set of labels,*
- *$\rightarrow \subseteq Q \times (\mathscr{P}(\Sigma) \setminus \{\emptyset\}) \times Q$ is a finite set of transitions (where $\mathscr{P}(\Sigma)$ denotes the powerset of Σ). A transition (q, L, q') is also denoted $q \xrightarrow{L} q'$.*

An execution trace is a finite sequence of labels in $\mathcal{L}^*$. ε denotes the empty sequence. The concatenation of two traces σ_1, σ_2 is denoted $\sigma_1.\sigma_2$.

Furthermore, we express security properties with LTL formulas, which concisely formalise them with the help of a small number of special logical operators and temporal operators [16]. Given a set of atomic propositions AP and $p \in AP$, LTL formulas are constructed by using the following grammar $\phi ::= p \mid (\phi) \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathbf{X}\phi \mid \phi_1\mathbf{U}\phi_2$. Additionally, a LTL formula can be constructed with the following operators, each of which is defined in terms of the previous ones:
$\top \overset{def}{=} p \vee \neg p$, $\bot \overset{def}{=} \neg\top$, $\phi_1 \wedge \phi_2 \overset{def}{=} \neg(\neg\phi 1 \vee \neg\phi_2)$, $\phi_1 \rightarrow \phi_2 \overset{def}{=} \neg\phi_1 \vee \phi_2$, $\mathbf{F}\phi \overset{def}{=} \top\mathbf{U}\phi$, $\mathbf{G}\phi \overset{def}{=} \neg\mathbf{F}(\neg\phi)$.

$SF(\phi)$ denotes the set of sub-formulas of ϕ. This set is is defined inductively as follows: $SF(p) \overset{def}{=} \{p\}$; $SF(\neg\phi) \overset{def}{=} \{\neg\phi\} \cup SF(\phi)$; $SF(\phi_1 \vee \phi_2) \overset{def}{=} \{\phi_1 \vee \phi_2\} \cup SF(\phi_1) \cup SF(\phi_2)$; $SF(\mathbf{X}\phi) \overset{def}{=} \{\mathbf{X}\phi\} \cup SF(\phi)$; $SF(\mathbf{U}\phi) \overset{def}{=} \{\mathbf{U}\phi\} \cup SF(\phi)$. When the other operators are used, their definitions allow to inductively recover the set of sub-formulas as well.

4.2 Property Type

We use property types to express general features, which are independent of the type of system under audit. A property type is a specialised LTL formula whose atomic propositions are predicates, a predicate being either an expression of one or more variables defined on some specific domains, or a nullary predicate, which corresponds to an atomic proposition.

Definition 2 (Property Type).

- *Pred denotes a set of predicates of the form P (nullary predicate) or $P(x_1, \dots, x_k)$ with $x_1, \dots, x_k$ some variables that belong to the set denoted X;*
- *The domain of a predicate variable $x \in X$ is written $Dom(x)$;*
- *A property type Φ is a LTL formula built up from predicates in Pred. $\mathscr{P}$ denotes the set of property types.*

Property types cannot be evaluated by model-checkers as they are composed of predicates. They require to be instantiated before, i.e. predicate variables have to assigned to values. The instantiation of a property type is called a property instance. It has the same LTL structure as its property type, but the instantiated predicates now form propositions.

Definition 3 (Property Instance). *A property instance ϕ of the property type Φ is a LTL formula resulting from the instantiation of the predicates of Φ.*

The function that instantiates a property type to one property instance, i.e. which associates each variable of the predicates to a value, is called a binding:

Definition 4 (Property Binding). *Let X' be a finite set of variables $\{x_1, \dots, x_k\} \subseteq X$. A binding is a function $b : X' \rightarrow Dom(X')$, with $Dom(X') = Dom(x_1) \times \dots \times Dom(x_k)$.*

Applying a binding to the variables of a property type gives a property instance related to this binding. For instance, applying $\{x \rightarrow pass := \text{"1234"}\}$ to $Sensitive(x) \rightarrow Encrypted(x)$ returns the property instance $Sensitive(pass := \text{"1234"}) \rightarrow Encrypted(pass := \text{"1234"})$.

The writing of property types is not a straightforward activity in the sense that it requires a good expert knowledge on LTL. This is why we provide a set of property types in Sect. 3.1. We recommend writing property types by firstly formulating general security concepts with predicates, and by applying or composing the LTL patterns given in [10] on those predicates. These patterns help structure LTL formulas with precise and correct statements that model common situations, e.g., the absence of events, or cause-effect relationships.

In order to reduce execution times, our algorithms take advantage of particular formula, which we call *conditional* property types. These will be useful to enhance the property type instantiation algorithm given in Sect. 4.5. Intuitively, a conditional property type is formed by an implication. It is worth noting that the definition given below identifies some conditional property types but is not exhaustive.

Definition 5 (Conditional Property Types).

1. *Φ is a conditional property type iff Φ is a property type of the form: $P \rightarrow Q$, $\mathbf{G}(P \rightarrow Q)$, $\mathbf{F}(P \rightarrow Q)$, $\mathbf{G}P \rightarrow Q$, $(P \rightarrow Q) \vee \Phi_2$, $\mathbf{G}((P \rightarrow Q) \vee \Phi_2)$, $\mathbf{F}((P \rightarrow Q) \vee \Phi_2)$ with P a predicate or a conjunction of predicates, Q a predicate or a disjunction of predicates, and Φ_2 an LTL formula;*
2. *$antecedent(\Phi)$ denotes the antecedent of the implication of Φ, which is either equal to P or $\mathbf{F}P$ here;*
3. *$consequent(\Phi) \overset{def}{=} Q$.*

We provide, in Sect. 3.1, 12 property types modelling the security measures of the ENISA organisation related to communications. These property types are made up of the predicates given in Table 2. For example, the security measure GP-TM-38 recommends to encrypts sensitive data, which is formulated as $\mathbf{G}(Sensitive(x) \rightarrow Encrypted(x))$. GP-TM-24 recommends encrypting authentication credentials. This measure is formulated as: $\mathbf{G}((LoginAttempt(c) \wedge Credential(x) \rightarrow Encrypted(x))$, which intuitively means that every time a component attempts to log in to another component c by using credentials x, then x must be encrypted. GP-TM-53 suggests that error messages must not expose sensitive information. This is formulated with $\mathbf{G}((ErrorResponse \vee \neg ValidResponse) \rightarrow (\neg(BlackListedWord(x)))$. This formula intuitively means that every HTTP response composed of an error message or having a status higher than 299 must not include blacklisted words. If we apply the binding $\{c \rightarrow c1, x \rightarrow login := toto\}$ on the second property type, we obtain the property instance $\mathbf{G}((LoginAttempt(c1) \wedge Credential(login := toto) \rightarrow Encrypted(login := toto))$.

4.3 MLCA Step 2: LTS Completion

Model-checkers cannot directly check the satisfiability of property types on LTSs as the properties are made up of predicate variables. This step prepares the property type instantiation by helping auditors complete the LTS transitions with some predicates of *Pred*, i.e. the predicates used to write the property types. Once this step is finished we obtain new LTSs that share the same alphabet as the property types. As an LTS encodes concrete behaviours, this step actually adds instantiated predicates of *Pred*, i.e. predicates whose variables are assigned to concrete values found in the LTS actions. Completing the LTS transitions with these instantiated predicates comes down to analysing/interpreting LTS transitions or paths and to add new labels on transitions to extend the LTS semantics. To performs this activity in an automatic manner, this step uses an expert system, made up of inference rules, which encode some expert knowledge about SUA. The expert system offers the benefits to automate the LTS transition completion and to save time by allowing to reuse it on several IoT systems.

We represent inference rules with this format: *When conditions on facts Then actions on facts* (format taken by the Drools inference engine [28]). The facts, which belong to the knowledge base of the expert system, are here the transitions of an LTS. To ensure that the transition completion is performed in a finite time and in a deterministic way, the inference rules have to meet these hypotheses:

- Finite complexity: a rule can only be applied a limited number of times on the same knowledge base,
- Soundness: the inference rules are Modus Ponens (simple implications that lead to sound facts if the original facts are true).

We devised 28 inference rules, which are available in [31]. These can be categorised as follows:

- Structural information: two rules are used to add the propositions "Begin" and "End", which describe the beginning and end of user sessions in LTS paths;
- Nature of the actions: 9 rules add information about the nature of the actions. Given a transition $q_1 \xrightarrow{\{a(\alpha)\}} q_2$, some rules analyse the parameter assignments in α and complete the transition with new propositions expressing that $a(\alpha)$ is a request or a response, an input or an output, the component that sent $a(\alpha)$, etc. Other rules analyse α to interpret if the action corresponds to an error response (analysis of the values in α to detect words like "error" or analysis of HTTP status, etc.). For instance, the first rule of Fig. 9 adds the proposition ValidResponse if the transition expresses a response whose HTTP status is between 200 and 299. The status inspection is performed by the procedure isOk();
- Security information: the other rules add predicates related to security on LTS transitions. For instance, we devised a rule that checks whether some parameters are encrypted. Other rules analyse the LTS paths to try recognise specific patterns (transition sequences containing specific words), e.g., authentication attempts, successful or failed authentications. For instance, the second rule of Fig. 9 detects a correct authentication. It adds the proposition "LoginAttempt(c)" on the transition labelled by the credentials and "Authenticated(c)" on the transition whose action encodes a correct authentication with the external component c.

```
rule "validResponse"
when
$t : Transition(isReq() == false,
isOk())
then
$t.addLabel("validResponse");
end
```

```
rule "Authentication"
when
$t1: Transition(isRequest(), contains("
login"), contains("password"))
$t2: Transition(isResponse(),isOK(),
sourceState = $t1.targetState)
then
$t1.addLabel("LoginAttempt("+
compoSender($t1) + ")");
$t2.addLabel("Authenticated("+
compoSender($t1) + ")");
end
```

Fig. 9. Inference rule examples [29].

Once the expert system has completed the LTS $\mathscr{L}(c1)$, we obtain a new LTS denoted $\mathscr{L}'(c1)$.

4.4 MLCA Step 3: LTS Verification

The notion of expert system, used in the previous step, suffers from some limitations as most of the current AI applications. One of them relates to the difficulty of knowledge acquisition. In other terms, the expert system rules may be incomplete or not adapted to SUA. For instance, the rule "Authentication" of Fig. 9 is based on the keywords "login" and "password" for recognising a successful authentication, but other keywords might be used instead. With this example, the expert system can be easily completed with new rules. But, some rules also depend of external tools, which might provide incorrect information. We observed this case for the rule dedicated to the recognition of data encryption. This rule relies on a tool, which often returns false positives or negatives, especially when the data length is short. This is why the auditors have to inspect the LTSs to assess the transition completion correctness. We propose two solutions to help auditors in this verification:

- the expert system notices and returns the names of the rules, which have not been triggered previously. The returned list of rules warns the auditor that some predicates have not been added on transitions;
- this step also automatically checks whether invariants hold on every LTS transitions. We provide a set of 7 invariants written with property types. The first 4 invariants are used to detect missing labels that have to be found on LTS transitions. The remaining ones are used to detect incorrect label completion on transitions. The predicate variables c and $c2$, which are used with some invariants, only refer to components. To make these invariants concrete, we assign both variables to the components that belong to the set C. Then, we check whether the invariants hold on the LTSs by calling a model-checker. When an invariant is not satisfied, the LTS transition is returned to the auditor, so that he/she can manually review it.

Table 3. Invariant used to verify the LTS transition completion correctness.

$Begin \wedge F End$
$\mathrm{G}(Input \vee Output)$
$\mathrm{G}(Request \vee (Response \wedge (ValidResponse \vee ErrorResponse)))$
$G((Request \wedge From(c) \wedge To(c2)- > F(Response \wedge To(c) \wedge From(c2))$
$\mathrm{G}(From(c) \rightarrow \neg To(c))$
$\mathrm{G}(\neg Input \wedge Output)$
$LoginAttempt(c) \rightarrow Request \wedge To(c)$

4.5 MLCA Step 4: Property Type Instantiation

Given an LTS $\mathscr{L}'(c1)$, this step aims at instantiating the property types of $\mathscr{P}$ to obtain a set of property instances that can be evaluated on the paths of $\mathscr{L}'(c1)$. We denote $\mathscr{P}(\mathscr{L}'(c1))$ this set of property instances. Intuitively, a property type Φ is instantiated with bindings, which assign values to the predicate variables of Φ. This step derives these bindings from the labels (instantiated predicates) added on the LTS transitions by the expert system previously.

We proposed in [29] a preliminary algorithm implementing the property type instantiation. To illustrate this algorithm, consider the property type $G(Sensitive(x) \rightarrow Encrypted\ (x))$ modelling the ENISA security measure GP-TM-38 and an LTS $\mathscr{L}'(c1)$ having the labels Sensitive(login:=toto), Encrypted(login:=toto), Encrypted(resp:=" HTTP/1.1 200 OK..."). The Property Type Instantiation algorithm builds the set $Dom(x) = \{login := toto, resp := "HTTP/1.1200OK..."\}$. Consequently, two bindings and two property instances are automatically built. Hence, despite the need for checking the LTS completion correctness, an auditor does not have to be an expert in LTL for using our approach.

We showed in [29] that the time complexity of the Property Type Instantiation algorithm is polynomial, but it is manifest that the LTS size and the number of predicate values found on the LTS transitions will increase the number of generated property instances and then negatively affect execution times, as verifying whether LTL formulas hold is usually time consuming.

From this statement, we now propose to focus on an enhancement of the algorithm given in [29] to lower the time complexity of MLCA. We indeed observed that our algorithm produces a set of pointless property instances in the sense that these formulas are always true. Consider the previous example of property type, that is $G(Sensitive(x) \rightarrow Encrypted(x))$ along with the LTS $\mathscr{L}'(c1)$ having the same labels *Sensitive(login:=toto)*, *Encrypted(login:=toto)*, *Encrypted(resp:="HTTP/1.1 200 OK...")*. The last label gives the binding $\{x \rightarrow resp := "HTTP/1.1\ 200\ OK..."\}$ and the property type $G(Sensitive(resp := "HTTP/1.1\ 200\ OK...") \rightarrow Encrypted(resp := "HTTP/1.1\ 200\ OK..."))$. The latter will always be true, as the antecedent part of the formula $Sensitive(resp := “...")$ will always be false (the predicate Sensitive(resp:="...") is not labelled on any transition of $\mathscr{L}'(c1)$). This statement can be observed with the conditional property types given in Definition 5. This is formulated by the following proposition:

Proposition 1. *Let Φ be a conditional property type such that there exist $P_1(x_1,\ldots,x_k) \in antecedent(\Phi)$ and $P_2(x_1,\ldots,x_k) \in consequent(\Phi)$. X' is the finite set* *of predicate variables of* Φ. *Let* $P_2(v_1,\ldots,v_k) := true$ *be an instantiated predicate of* $P_2(x_1,\ldots,x_k)$ *and* $b : X' \to Dom(X')$ *such that* $b' : \{x_1 \to v_1,\ldots,x_k \to v_k\}$ *is a restriction of b.*

The property instance $\varphi := b(\Phi)$ *resulting from the instantiation of* Φ *with b is always true.*

Sketch of Proof of Proposition 1: let us consider the conditional property type $\Phi := P \to Q$ with P a predicate or a conjunction of predicates, Q a predicate or a disjunction of predicates.

The property instance φ derived from Φ with b is denoted $\varphi := \varphi_1 \to \varphi_2$.
$\varphi_2 := P_2(v_1,\ldots,v_k)$ or $\varphi_2 := P_2(v_1,\ldots,v_k) \vee \varphi_2'$ with φ_2' an LTL formula, such that $P_2(x_1,\ldots,x_k) \in SF(Q)$ and $b' : \{x_1 \to v_1,\ldots,x_k \to v_k\}$ is the restriction of b over $\{x_1,\ldots,x_k\}$.

As a consequence, as $P_2(v_1,\ldots,v_k)$ is true, φ_2 is also true. As φ_2 is true, $\varphi_1 \to \varphi_2$ is true whatever the evaluation of φ_1. Finally, $\varphi := b(\Phi)$ is always true. The same reasoning can be applied on the other conditional property types of Definition 5. ■

In reference to Proposition 1, we now provide a new implementation of the property type instantiation in Algorithm 1. The main difference between this algorithm and the previous one lies in lines 4–6. It begins by storing the predicates of the property type Φ in a set *PS*, which contains the predicates from which bindings will be computed (in contrast to the previous algorithm where all the predicates were considered). Now, if Φ is recognised as a conditional property type, if it includes two predicates $P1$, $P2$ having the same variables and if $P1$ belongs to the antecedent part of Φ whereas $P2$ belongs to the consequent part, then $P2$ is removed from *PS*. Removing $P2$ from *PS* allows to ignore the predicate values leading to pointless property type instances, as in our example. In lines 7–10, Algorithm 1 covers the LTS transitions and every label $P(v_1,\ldots,v_k)$ to generate domains of values on condition that the related predicate $P(x_1,\ldots,x_k)$ belong to *PS*. Afterwards, the property types are instantiated as before.

The time complexity of Algorithm 1 is proportional to $|\mathscr{P}|(p^2 + | \to |p + |D|)$. This complexity is higher than the one of the first algorithm, but it should return less property instances. As a consequence, MLCA should verify less property instances and hence consume less time with this algorithm. This is confirmed in our evaluation results (Sect. 5).

4.6 MLCA Step 5: Property Instance Verification

Given an LTS $\mathscr{L}'(c1)$, this step comes down to calling a model-checker for verifying whether $\mathscr{L}'(c1)$ satisfies the property instances of $\mathscr{P}(\mathscr{L}'(c1))$. If the model-checker detects a counterexample path that violates a property instance $\phi \in \mathscr{P}(\mathscr{L}'(c1))$, our approach reports the security measure related to ϕ, which is not (correctly) implemented. Our approach also returns the counterexample so that auditors may analyse it to comprehend the captured failure scenario.

Algorithm 1. Property Type Instantiation 2.

```
input  : LTS $\mathscr{L}'(c1)$, property type set $\mathscr{P}$
output: Property instance set $\mathscr{P}(\mathscr{L}'(c1))$
1  Compute $Dom(deps)$ from $Dg(c1)$;
2  $X' := \emptyset$;
3  foreach $\Phi \in \mathscr{P}$ do
4      $PS$ set of of predicates of $\Phi$;
5      if $\Phi$ is a conditional property type and $\exists P1(x_1,\dots,x_k), P2(x_1,\dots,x_k) \in PS : P1(x_1,\dots,x_k) \in$
       $SF(antecedant(\Phi)) \wedge P2(x_1,\dots,x_k) \in SF(consequent(\Phi))$ then
6          $PS := PS \setminus \{P2(x_1,\dots,x_k)\}$;
7      foreach $l \in L$ with $s_1 \xrightarrow{L} s_2 \in \rightarrow_{\mathscr{L}'(c1)}$ do
8          if $l = P(v_1,\dots,v_k)$ and $P(x_1,\dots,x_k) \in PS$ then
9              $Dom(x_i) = Dom(x_i) \cup \{v_i\} (1 \le i \le k)$;
10             $X' := X' \cup \{x_1,\dots,x_k\}$;
11     if $X'$ is not equal to the set of predicate variables of $\Phi$ then
12         return a warning;
13     else
14         $D := Dom(x_1) \times \cdots \times Dom(x_n)$ with $X' = \{x_1,\dots,x_n\}$;
15         foreach binding $b \in D^{X'}$ do
16             $\phi := b(\Phi)$;
17             $\mathscr{P}(\mathscr{L}'(c1)) = \mathscr{P}(\mathscr{L}'(c1)) \cup \{\phi\}$;
```

4.7 Limitations

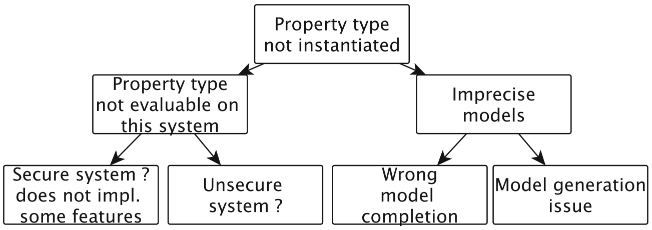
Property type not instantiated
Property type not evaluable on this system
Imprecise models
Secure system ? does not impl. some features
Unsecure system ?
Wrong model completion
Model generation issue

Fig. 10. Some MLCA limitations.

Our approach suffers from some limitations. The first is related to the need for formulating security measures with property types. Even though we propose a substantial set of property types, auditors might need further formulas for their own use. But, writing LTL formulas requires some expertise and experience.

The generalisation of our approach is also restricted by the requirements A1-A3. In particular, the event logs must be formatted by means of regular expressions so that the action types can be identified. Although we have observed that this task is not too difficult to carry out on HTTP messages, it is manifest that this is not generalisable to any kind of protocols, especially those encrypting some parts of the message content.

We also have showed that Algorithm 1 may return warnings showing that property types cannot be instantiated. Figure 10 summarises the main reasons. The two first ones (left side of the figure) concern SUA: it might not implement some features. Here, the auditor has to establish and decide if SUA is secure even though a security feature is missing. Additionally, a property type may not be instantiated on account of the models (right side of the figure). On the one hand, the LTSs are generated by a model learning algorithm, which may infer under-approximated (rejection of valid behaviours) or over-approximated (acceptance of invalid behaviours) models. It is worth noting that the model learning tools available in the literature tend to generate more and more precise models, which accept most of legal behaviours and reject most of anomalous ones [30]. On the other hand, the LTSs may be incorrectly completed by the second step of MLCA. As stated earlier, the resulting LTSs should be manually reviewed.

4.8 Implementation

Our approach is implemented as a tool chain, which gathers three tools: CkTail, which implements a model learning approach specialised for communicating systems, SMV-maker, which completes the LTSs produced by CkTail by means of an expert system and instantiates property types, and the model-checker NuSMV. The tools, source codes, examples of property types and traces are available in [31]. These tools were employed to evaluate our approach with several case studies.

5 Empirical Evaluation

The experiments presented in this section aim to assess the capabilities of our algorithms to verify whether security measures are active on models inferred from logs in terms of precision (exact detection of active and inactive security measures) and performance. This section supplements our preliminary evaluation proposed in [29] as we study the capabilities of our algorithms through 3 research questions and as we consider more systems. These research questions are:

– RQ1: is MLCA able to detect active security measures? RQ1 investigates the sensitivity of MLCA, that is its capability to identify the security measures related to communications that are truly implemented on IoT systems;
– RQ2: is MLCA able to detect inactive security measures? RQ2 investigates the specificity of MLCA, that is its capability to uncover the security measures that are not considered or incorrectly implemented;
– RQ3: how long does MLCA take to verify whether security measures are correctly implemented? RQ3 studies the performance of our algorithms and how they scale with the size of the event logs, and with the number of the property types.

5.1 Empirical Setup

This study was conducted on five IoT systems integrating varied devices, gateways and external cloud servers communicating over HTTP. We assembled and configured these systems using the available security options. From these systems, we collected event logs composed of HTTP messages on gateways or servers.

- **S1** is composed of 3 motion sensors that turn on a light-bulb when they detect movements. They communicate through a gateway; all of them have user interfaces, which may be used for configuration. The main purpose of this system is to focus on classical attacks (code injection, brute force, availability) and on the related security measures GP-TM 22, 25, 38, 52, 53;
- **S2** includes a motion sensor, a switch and an IP camera. The motion sensor communicates with the switch in clear-text, and with the camera with encrypted messages. The camera also communicates with external clouds requiring authentication and encryption. In this system, we mainly focus on security measures related to encryption and authentication, GP-TM 22, 24, 25, 26, 38, 42, 53;
- **S3** is composed of 3 IP cameras, which communicate with external cloud servers. Two cameras can update their software remotely. This system aims at focusing on security measures related to update mechanisms, GP-TM 18, 19, 38, 48, 53;
- **S4** is made up of a temperature sensor, which communicates with a gateway. A user can connect to the gateway for reading temperature curves. We devised this system so that the gateway is vulnerable to XSS code injections. Here, we mainly focus on security measures related to encryption, authentication, and code injection, GP-TM 22, 24, 25, 26, 38, 42, 52, 53;
- **S5** is composed of an IP camera, which interacts with a NTP, a SMTP, and an FTP server. The camera authenticates to the FTP server (with encrypted credentials), and to the SMTP server (with unencrypted credentials). A user can connect to the camera to get images or videos. With this system, we mainly focus on security measures related to encryption and authentication, GP-TM 22, 24, 26, 38, 42, 52, 53.

5.2 RQ1: Is MLCA Able to Detect Active Security Measures?

To answer RQ1, we initially experimented the above systems by hands. We gathered a first set of event logs that capture "normal behaviours" and generated first models with CkTail. From these, we identified the testable components, on which we applied a set of penetration testing tools specialised to Web applications. During this testing stage, we collected larger event logs. From them, we generated the final models used for the evaluation.

Besides, we manually analysed the source codes of the software used by the devices (sensors, gateways) when available and the event logs to list the ENISA security measures that are active and those that are not (correctly) implemented . This task allowed us to compare the results of MLCA with our observations and hence to measure the sensitivity and specificity of our approach.

For this evaluation, we considered the 12 property types given in Table 1. We ran MLCA to generate the LTSs and property instances from these property types. To not perform an unbiased evaluation, we considered two cases: the case where we did not manually completed the generated LTSs denoted "w/o modification" and the case where we manually completed them as required in the third step of the approach, denoted "w/ modifications". And finally, we called the model-checker NuSMV to check whether the property instances hold on the LTSs.

For this research question, we measured the sensitivities of MLCA (rate of correct property instances that evaluate to true) using Algorithm 1 "w/o modification" and "w/

modifications". The comparison of our previous MLCA algorithms given in [29] cannot be based on sensitivity measurements as both approaches generate different sets of property instances (in general terms, the numbers of conditional positives are different). To compare both algorithms, we measured their number of false positives FP, i.e. the number of unsatisfied property instances, which should hold.

Results:

Table 4. False positives (FP) and Sensitivity of MLCA.

IoT syst	#instantiated property types/total	FP w/o modif. MLCA of [29]	FP w/o modif. MLCA	Sensitivity w/o modif. MLCA	Sensitivity w/ modif. MLCA
S1	44/60	3	3	97.5%	98.33%
S2	14/36	0	0	100%	100%
S3	19/36	0	0	100%	100%
S4	13/24	0	0	100%	100%
S5	26/48	0	0	100%	100%

Table 4 summarises the results of our experiments. Col. 2 gives the ratio of property types that have been instantiated over the property types available for all the components of a system. For instance, S1 has 5 components, hence 5*12 property types can be used. 44 property types were instantiated for this system. None of the systems allowed to instantiate all the property types. This result was expected as none of the systems implement all the security features considered in the security measures, e.g., password recovery for S1, or authentication for some components of S2, S3. Then, Col. 3 and 4 give the number of false positives observed with MLCA of [29] and the MLCA algorithms presented in this paper. The two last columns provide the sensitivity of MLCA without or with manual modifications of the LTSs (after the third step of MLCA).

Firstly, col. 3 and 4 confirm that our new property instance algorithm (Algorithm 1) does not change the number of false positives. Col. 5 shows that the overall sensitivity of MLCA "w/o modification" is 99,5%. When we manually complete the LTSs the overall sensitivity increases up to 99,66 %. The sensitivity is below 100 % with S1. After inspection, we observed that the difference of sensitivity "w/o modification" and "w/ modifications" with S1 comes from an inference rule of the expert system, which tries to recognise whether parameters are encrypted. This rule computes the message entropy, but this technique is sometimes not precise enough to recognise encryption. Hence some LTS transitions were incorrectly completed with predicates of the form Encrypted(x). We did not find any better solution at the moment. Still with S1, the sensitivity "w/ modifications" remains below 100% on account of a false positive related to the security measure GP-TM-25. We observed that this measure is detected as active on account of an over-generalisation of the LTSs. Apart from this issue related to model learning, these results tend to show that MLCA is very effective to check whether security measures are correctly implemented since the sensitivity of MLCA is close to 100%.

Table 5. Specificity of MLCA.

IoT syst	#instantiated property types /total	Specificity w/o modif. MLCA of [29]	Specificity w/o modif. MLCA	Specificity w/ modif. MLCA
S1	44/60	99.07%	99.07%	99.07%
S2	14/36	91.66%	91.66%	91.66%
S3	19/36	92.86%	92.86%	92.86%
S4	13/24	100%	100%	100%
S5	26/48	67.57%	67.57%	89.19%

5.3 RQ2: Is MLCA Able to Detect Inactive Security Measures?

We measured the specificity of our approach, that is the rate of inactive security measures that are correctly identified by MLCA. We recall that a security measure is detected as inactive for a given system when a property instance derived from this security measure does not hold on an LTS. We considered the same models generated for RQ1 along with the same property types. In the same way, to not perform an unbiased evaluation, we considered the two cases "w/o modification" and "w/ modifications".

Results: Table 5 shows the specificity measurements of our algorithms. The number of instantiated property types and the number of property instances given in Col. 2 remain the same. Col. 3 and 4 compare the specificity of MLCA of [29] with the algorithms presented in this paper. Next, Col. 4 and 5 allow to compare the specificity without or with manual modifications of the LTSs.

The overall specificity without modification of the LTSs is 91.8% with both algorithms. Again, this confirms that the property instance optimisation used in Algorithm 1 does not modify the results of MLCA. We observe that MLCA returns some false negatives, most of them being observed with S5. After inspection, we observed that most of these false negatives come again from the expert system rule related to encryption recognition. Here, this rule has detected that many encrypted messages are not encrypted. After the manual modification of the LTSs, the specificity reaches 96,12%. The remaining false negatives come from some LTS under-approximations, i.e. rejections by the LTSs of some correct behaviours.

These results tend to show that MLCA is very effective for detecting inactive security measures since the overall specificity of MLCA is higher than 91% without modification of the models. We believe that the benefits of automatically learning models and instantiating LTL formulas exceed the drawback of having a few false negatives.

5.4 RQ3: How Long Does MLCA Take to Verify Whether Security Measures Are Correctly Implemented?

To answer RQ3, we performed three experiments. We firstly studied how the tool scales with the size of the event logs. We collected 20 new event logs from S5 by varying the number of events between 500 to 100000, then we measured execution times. We

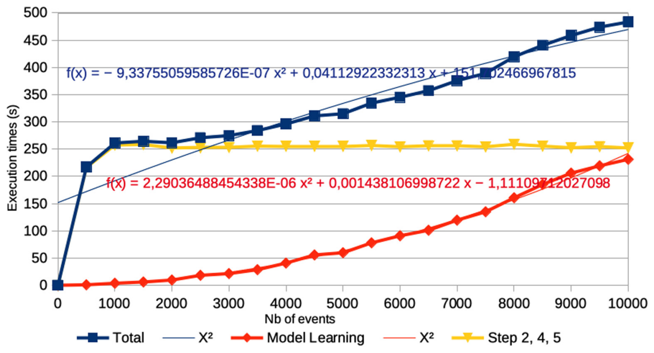

Fig. 11. Execution times vs. number of events.

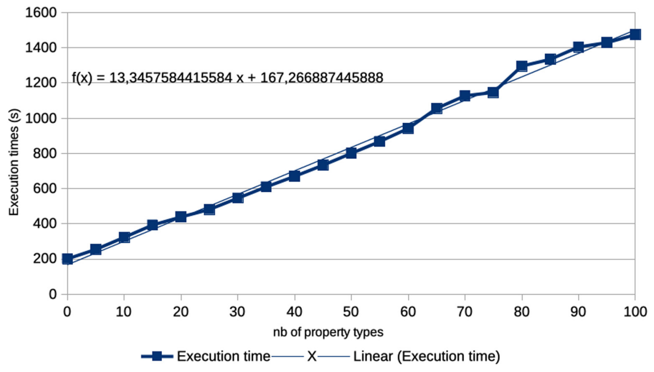

Fig. 12. Execution times vs. number of property types.

also studied how the tool scales with the number of property types. We measured the execution times of MLCA by using the event log of S5 and by increasing the property type number from 0 to 100. Finally, we measured the execution times obtained with the MLCA of [29] and the MLCA algorithms proposed in this paper on the 5 IoT systems. We also collected the number of generated property instances with both algorithms. These Experiments were carried out on a computer with 1 Intel® CPU i5-6500 @ 3.2GHz and 32GB RAM.

Results: Figure 11 depicts, with the curve Total, execution times in seconds w.r.t. the event log sizes. We observe that MLCA returns results within reasonable time even with large event logs. But, the curve follows a quadratic regression and reveals that our tool does not scale well. To understand this problem, we completed the figure with two additional curves depicting the times required to learn models and the times consumed by the other steps of the approach (Steps 2–5). These two curves clearly show that the scaling problem comes from the model learning algorithm CkTail.

Figure 12 depicts execution times in seconds w.r.t. the number of property types given as inputs. We observe that the curve follows a linear regression, showing that the tool scales well according to the number of property types. Even with 100 property types, execution times remain reasonable.

Table 6. Number of property instances and execution times measured on the 5 IoT systems.

IoT Syst	#property instance MLCA of [29]	#property instance MLCA	Execution times MLCA of [29]	Execution times Algorithm 1
*S*1	823	334	31708	4533
*S*2	37	37	509	510
*S*3	39	39	165	162
*S*4	80	68	76	60
*S*5	128	88	372	252

Table 6 compares the number of generated property instances along with execution times in seconds for the five IoT systems between the MLCA of [29] and the algorithms we proposed in this paper. In average, we observe that the new MLCA algorithms help reduce execution times by 28% thanks to the optimisations performed to reduce the set of generated property instances. With some systems (S2 and S3), we observe that both algorithms return the same amount of property instances and around the same execution times. With these systems, the property types that are instantiated are not conditional property types. In contrast, we observe that using Algorithm 1 with S1 allows to strongly reduce the set of property instances and to lower the execution time by 85%.

These results tend to show that MLCA can be used in practice even with large event logs, but it suffers from insufficient scalability, on account of the model learning step. These results also confirm that Algorithm 1 allows to save execution time.

5.5 Threat to Validity

Some threats to validity can be identified in our evaluation. The first factor, which may threaten external validity, applies to the case studies. Most of them are IoT systems using HTTP. But many communicating systems rely on other kinds of protocols, from which it may be more difficult to identify senders, receivers, requests or responses. Hence, we cannot conclude that our results are generalisable to IoT systems. This is why we deliberately avoid drawing any general conclusion. This threat is somewhat mitigated by the fact that HTTP is used by numerous communicating systems. However, the HTTP traffic should also be decipherable so that MLCA can recognise behavioural patterns and add predicates on LTS transitions. It is manifest that more experimentations are required on further kinds of systems.

There are also some threats to internal validity. The quality of the models produced by the first step of the approach strongly depends on the size and the formatting of the event logs. We showed in the evaluation that we can obtain some false positives / negatives on account of the lack of precision of the models. At the moment, none of the passive model learning tool generates "exact" models, they usually are slightly under- or over-approximated. We also mentioned that the outcomes of MLCA depend on the manual LTS verification carried out in Step 3. It is manifest that the outcomes of this step vary in accordance with the auditor knowledge and expertise. To avoid this potential threat, we considered two cases: no modification of the LTSs and manual modification. The first case gives the worst sensitivity and specificity of MLCA on the five IoT systems considered for the evaluation.

6 Conclusion

This paper has proposed the design and evaluation of an approach called MLCA (Model-Learning-Checking Approach), which combines model learning and model checking to help audit the security of IoT systems. It requires an event log along with security properties modelled with property types. The latter are generic LTL formulas, which can be used independently of the IoT system under audit. We provide 12 property types expressing some security measures provided by the ENISA organisation. MLCA automatically generates models from the event log, then it assists auditors in the generation of concrete properties by instantiating property types. This instantiation is carried out by an expert system made up of inference rules, which encode some expert knowledge about the kind of system under audit. Our evaluation, performed on 5 IoT systems, showed that MLCA is effective in detecting security issues, and that it can be used in practice on large event logs.

Nevertheless, several aspects need to be investigated in the future. The model learning approach requires some assumptions to generate precise models. We will investigate if some of them could be relaxed so that more systems under audit could be considered. Our evaluation showed that the use of an expert system offers a great potential for instantiating property types, especially when auditors are non-expert LTL users. However, this benefit strongly depends on the successful implementation of the expert system rules. We indeed observed in the experimentations that a few property types were not completely instantiated on account of the lack of precision of some rules. This is why the property type instantiation and LTS completion steps require a review to detect potential issues. Finding a way to get rid of this manual step is hence another direction for future work.

Acknowledgement. Research supported by the French Project VASOC (Auvergne-Rhône-Alpes Region) https://vasoc.limos.fr/.

References

1. Ahmad, A., Bouquet, F., Fourneret, E., Le Gall, F., Legeard, B.: Model-based testing as a service for iot platforms. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 727–742. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_55
2. Beschastnikh, I., Brun, Y., Abrahamson, J., Ernst, M.D., Krishnamurthy, A.: Using declarative specification to improve the understanding, extensibility, and comparison of model-inference algorithms. IEEE Trans. Softw. Eng. **41**(4), 408–428 (2015). https://doi.org/10.1109/TSE.2014.2369047

3. Beschastnikh, I., Brun, Y., Ernst, M.D., Krishnamurthy, A.: Inferring models of concurrent systems from logs of their behavior with Csight. In: Proceedings of the 36th International Conference on Software Engineering, pp. 468–479. ICSE 2014, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2568225.2568246, http://doi.acm.org/10.1145/2568225.2568246
4. van der Bijl, M., Rensink, A., Tretmans, J.: Compositional testing with iOCO. In: Petrenko, A., Ulrich, A. (eds.) FATES 2003. LNCS, vol. 2931, pp. 86–100. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24617-6_7
5. Celik, Z.B., McDaniel, P., Tan, G.: Soteria: automated iot safety and security analysis. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18), pp. 147–158. USENIX Association, Boston, MA (July 2018). https://www.usenix.org/conference/atc18/presentation/celik
6. Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., Faruki, P.: Network intrusion detection for iot security based on learning techniques. IEEE Commun. Surv. Tutor. **21**(3), 2671–2701 (2019). https://doi.org/10.1109/COMST.2019.2896380
7. Cimatti, A., et al.: NuSMV 2: an opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_29
8. Costin, A., Zaddach, J.: Iot malware: Comprehensive survey, analysis framework and case studies (2018)
9. CSA: Security guidance for early adopters of the internet of things (iot), cloud security alliance, white paper (2015)
10. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002), pp. 411–420 (May 1999). https://doi.org/10.1145/302405.302672
11. ENISA: Baseline security recommendations for iot in the context of critical information infrastructures, Technical report (2017). https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot
12. ETSI: Methods for testing & specification; risk-based security assessment and testing methodologies, Technical report (2015). https://www.etsi.org/
13. Falcone, Y., Jaber, M., Nguyen, T.H., Bozga, M., Bensalem, S.: Runtime verification of component-based systems. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM 2011 - Proceedings of the 9th International Conference on Software Engineering and Formal Methods. Lecture Notes in Computer Science (LNCS), vol. 7041, pp. 204–220. Springer, Montevideo, Uruguay, November 2011. https://doi.org/10.1007/978-3-642-24690-6_15, https://hal.archives-ouvertes.fr/hal-00642969
14. Ge, M., Hong, J.B., Guttmann, W., Kim, D.S.: A framework for automating security analysis of the internet of things. J. Netw. Comput. Appl. **83**, 12–27 (2017). https://doi.org/10.1016/j.jnca.2017.01.033, http://www.sciencedirect.com/science/article/pii/S1084804517300541
15. Gutiérrez-Madroñal, L., La Blunda, L., Wagner, M.F., Medina-Bulo, I.: Test event generation for a fall-detection iot system. IEEE Internet Things J. **6**(4), 6642–6651 (2019). https://doi.org/10.1109/JIOT.2019.2909434
16. Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual, 1st edn., Addison-Wesley Professional, Boston (2011)
17. ISO: Iso/iec 27030 information technology - security techniques - guidelines for security and privacy in internet of things (iot) (2019)
18. Khan, M.A., Salah, K.: Iot security: review, blockchain solutions, and open challenges. Future Gener. Comput. Syst. **82**, 395–411 (2018). https://doi.org/10.1016/j.future.2017.11.022, http://www.sciencedirect.com/science/article/pii/S0167739X17315765

19. Lally, G., Sgandurra, D.: Towards a framework for testing the security of iot devices consistently. In: Saracino, A., Mori, P. (eds.) Emerging Technologies for Authorization and Authentication, pp. 88–102. Springer International Publishing, Cham (2018)
20. Maksymyuk, T., Dumych, S., Brych, M., Satria, D., Jo, M.: An iot based monitoring framework for software defined 5g mobile networks. In: Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication. IMCOM 2017, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3022227.3022331
21. Mariani, L., Pastore, F.: Automated identification of failure causes in system logs. In: Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on Software Reliability Engineering (ISSRE), pp. 117–126, November 2008. https://doi.org/10.1109/ISSRE.2008.48
22. Matheu-García, S.N., Ramos, J.L.H., Gómez-Skarmeta, A.F., Baldini, G.: Risk-based automated assessment and testing for the cybersecurity certification and labelling of iot devices. Comput. Stand. Interfaces **62**, 64–83 (2019)
23. Matheu Garcia, S.N., Hernández-Ramos, J., Skarmeta, A.: Toward a cybersecurity certification framework for the internet of things. IEEE Secur. Priv. **17**, 66–76 (2019). https://doi.org/10.1109/MSEC.2019.2904475
24. Mohsin, M., Anwar, Z., Husari, G., Al-Shaer, E., Rahman, M.A.: Iotsat: A formal framework for security analysis of the internet of things (iot). In: 2016 IEEE Conference on Communications and Network Security (CNS), pp. 180–188, October 2016. https://doi.org/10.1109/CNS.2016.7860484
25. Nadir, I., et al.: An auditing framework for vulnerability analysis of iot system, pp. 39–47 (June 2019). https://doi.org/10.1109/EuroSPW.2019.00011
26. NIST: Framework for improving critical infrastructure cybersecurity, version 1.1, standards and technology, Technical report (2018). https://doi.org/10.6028
27. OWASP: Owasp testing guide v3.0 project (2020). http://www.owasp.org/index.php/Category:OWASP_Testing_Project#OWASP_Testing_Guide_v3
28. "Red-Hat-Software": The business rule management system drools, March 2020. https://www.drools.org/
29. Salva, S., Blot, E.: Verifying the application of security measures in iot software systems with model learning. In: van Sinderen, M., Fill, H., Maciaszek, L.A. (eds.) Proceedings of the 15th International Conference on Software Technologies, pp. 350–360, ICSOFT 2020, Lieusaint, ScitePress, Paris, France, 7–9 July 2020. https://doi.org/10.5220/0009872103500360
30. Salva, S., Blot, E.: Cktail: model learning of communicating systems. In: Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2020, Prague, CZECH REPUBLIC, 5–6 May 2020
31. Salva, S., Blot, E.: Verifying the application of security measures in iot software systems with model learning, companion site (2020). https://perso.limos.fr/~sesalva/tools/mlc/. Accessed Oct 2020
32. Siby, S., Maiti, R.R., Tippenhauer, N.O.: Iotscanner: Detecting and classifying privacy threats in iot neighborhoods. CoRR abs/1701.05007 (2017). http://arxiv.org/abs/1701.05007
33. Wilson, J., Wahby, R., Corrigan-Gibbs, H., Boneh, D., Levis, P., Winstein, K.: Trust but verify: Auditing the secure internet of things, pp. 464–474 (July 2017). https://doi.org/10.1145/3081333.3081342
34. Zhang, Z.K., Cho, M.C.Y., Shieh, S.: Emerging security threats and countermeasures in iot. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, pp. 1–6. ASIA CCS15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2714576.2737091