
Moez Krichen\textsuperscript{1,2} and Roobaea Alroobaea\textsuperscript{3}

\textsuperscript{1}Faculty of CSIT, Al-Baha University, Saudi Arabia
\textsuperscript{2}ReDCAD Laboratory, University of Sfax, Tunisia
\textsuperscript{3}College of CIT, Taif University, Saudi Arabia

Keywords: Model-based, Testing, Security, Internet of Things, IoT, Smart Cities, Attack Tree, Price Timed Automaton, UPPAALL, TTCN-3, Cloud.

Abstract: In this paper we propose a new model-based framework for testing security properties of Internet of Things in Smart Cities. In general a model-based approach consists in extracting test cases from a formal specification either of the system under test or the environment of the considered system in an automatic fashion. Our framework is mainly built on the use of two formalisms namely Attack Trees and Price Timed Automata. An attack tree allows to describe the strategy adopted by the malicious party which intends to violate the security of the considered IoT system. An attack tree is translated into a network of price timed automata. The product of the constructed price timed automata is then computed using the well known UPPAALL platform. The obtained timed automata product serves as input for the adopted test generation algorithm. Moreover our framework takes advantage of the use of the standardized specification and execution testing language TTCN-3. With this respect, the obtained abstract tests are translated into the TTCN-3 format. Finally we propose a cloud-oriented architecture in order to ensure test execution and to collect the generated verdicts.

1 INTRODUCTION

Nowadays Internet of Things (IoT) is playing an important role in our modern society as a technology which allows to connect everyday objects to Internet. These objects are equipped with sophisticated interfaces which give them the capabilities to measure physical aspects from the environment and to interact with other entities by exchanging specific messages.

This new technology has provided a wide generation of valuable and innovative services. In this manner, modern cities are becoming smarter by adopting intelligent systems for water management, traffic control, energy management, street lighting, public transport, etc. However, these services can massively be attacked and compromised by several malicious parties whenever adequate and appropriate security measures are absent.

A few years ago, the smart devices that make up the Internet of Things, such as light bulbs, thermostats, webcams, and many others, were seen as potential targets for attackers to activate or disable remote devices and to harm consumers. Today, the IoT no longer represents a mere target, but a real platform which may be used by attackers to launch dangerous remote aggressions and to cause very serious incidents.

With the emergence of wide-ranging Open Source worms, such as Mirai (Antonakakis et al., 2017), capable of spreading to tens of millions of IoT devices, attackers can exploit these systems to generate a massive influx of traffic and disconnect any company or institution from Internet. Beyond the massive attack of the network, these IoT attack platforms can present other forms of threats, such as information theft and passwords decryption.

The attacks on industrial control systems have taken a disturbing turn. Cyber criminals attack the operational core of vital infrastructures taking advantage of their vulnerability. Recent attacks have not only disrupted the provision of essential services, such as electricity, but have also damaged automation systems to return to normal operation.

For instance the 2015 and 2016 attacks in Ukraine (Boyte, 2017) that caused power outages were perfectly planned and coordinated. The attackers managed to hijack the automation systems to cause power outages and then perform a well-ordered
The root of an AT corresponds to the goal the attacker aims to fulfill. The children of a node in the AT are refinements of the goal of the corresponding parent node into sub-goals. The refinement of an internal node of an AT can be either conjunctive or disjunctive:

- A conjunctive refinement is used when the fulfillment of all the children’s goals is needed to fulfill the parent’s goal. In this case we associate an AND-Gate with considered parent node (See Figure 1-(a)).
- A disjunctive refinement is used when the fulfillment of one of the children’s goals is enough to fulfill the parent’s goal. In this second case an OR-gate is associated with the considered node (See Figure 1-(b)).

\[\text{Figure 1: Different possible gates of an attack tree.}\]

We consider a finite set of attribute variables \(\text{Attris} = \{\text{Att}_1, \cdots, \text{Att}_n\}\). These attributes are used to describe the characteristics of the attacker like available resources for instance. We denote by \(\text{Vals} \subseteq (\mathbb{R}_{\geq 0})^n\) the set of valuations of the attributes.

The leaves of the AT correspond to the elementary actions the attacker has to execute. They are called basic attack steps (BAS). Each BAS is equipped with an additional attribute \(\text{Time}\) (which measures the time since the basic attack step started) and two preconditions:

\[\text{Ready2Start} : \text{Vals} \rightarrow \{0, 1\}\]
which indicates that the BAS can be started or not (i.e., has enough resources to start for example); and

\[\text{Able2Succeed} : \text{Vals} \rightarrow \{0, 1\}\]
which indicates whether the BAS can succeed or not. These preconditions are Boolean combinations of linear equations over Attributes. Moreover each BAS has an update function:

\[\text{Modify} : \text{Vals} \times \mathbb{R}_{\geq 0} \rightarrow \text{Vals}\]
which updates the attribute values when time elapses. At this level we assume that time dependence is linear between the attributes \(\{\text{Att}_1, \cdots, \text{Att}_n\}\) and the special attribute \(\text{Time}\).
Let $\text{PrePost}(\text{Attribs})$ be the set of all possible triples $(\text{ReadyToStart}, \text{AbleToSucceed}, \text{Modify})$ defined with respect to $\text{Attribs}$. We also define an Attacker Initializer which gives the initial valuation of the attributes. It is defined as:

$$\text{Init} : \{\text{Att}_1, \cdots, \text{Att}_n\} \rightarrow \mathbb{R}_{\geq 0}.$$  

The set of attack tree gate types is defined as:

$$\text{Gates} = \{\text{AND}, \text{OR}\}.$$  

An attack tree $A$ is formally defined as a tuple $(Nds, \text{Child}, \text{Rt}, \text{Attribs}, \text{Init}, \text{Ftrs})$ where:

- $Nds$ is a finite set of attacker nodes;
- $\text{Child} : Nds \rightarrow Nds^*$ associates a set of children to each parent node;
- $\text{Rt}$ corresponds to the root of the AT $A$ which defines the global goal of the attacker;
- $\text{Attribs}$ corresponds to the set of attributes of the AT $A$;
- $\text{Init}$ corresponds to the attacker initializer;
- $\text{Ftrs} : Nds \rightarrow \text{Gates} \cup \text{PrePost}(\text{Attribs})$ associates an AND/OR Gate with each internal node and a tuple $(\text{ReadyToStart}, \text{AbleToSucceed}, \text{Modify})$ with each leaf of the AT.

An example of an AT is given in Figure 2. This AT is inspired from the work of (Kumar et al., 2015). The goal of the attacker here is to crack the password of a protected file. As indicated by the figure the global goal of the attacker can be achieved by:

- Either cracking the password: this sub-goal can in turn be achieved using one of three possible choices (namely: Dictionary, Guessing or Brute Force attacks).
- Or performing a password attack: this sub-goal can be either achieved by a Social Engineering or a Key Logger attacks. The Social Engineering attack is in turn decomposed into two BASs namely: Generic Reconnaissance and Trap Execution. Similarly the Key logger attack is achieved within two BAS: Key Logger Installation and Password Intercept.

More details about this example can be found in the previously mentioned article (Kumar et al., 2015).

3 PRICED TIMED AUTOMATA

The model of Priced timed automata (PTA) (Behrmann et al., 2005) is an extension of timed automata, obtained by assigning costs to actions and locations. Next, we will denote by $\Psi(Y)$ the set of all possible Boolean predicates over a set $Y$ of continuous variables.

A priced timed automaton $P$ is defined as a tuple $(\text{Loc}, \text{loc}_0, \text{Cl}, \text{Act}, \text{Edg}, \text{Inv}, \text{Cost})$ where:

- $\text{Loc}$ is a finite set of states;
- $\text{loc}_0 \in \text{L}$ is the initial state;
- $\text{Cl}$ is a finite set of clocks;
- $\text{Act}$ is finite a set of labels;
- $\text{Edg} \subseteq \text{Loc} \times \Psi(\text{Cl}) \times \text{Act} \times 2^\text{Cl} \times \text{Loc}$ gives the set of transitions;
- $\text{Inv} : \text{Loc} \rightarrow \Psi(\text{Cl})$ assigns invariants to locations;
- $\text{Cost} : \text{Loc} \cup \text{Edg} \rightarrow \mathbb{N}_{\geq 0}^\text{Cl}$ assigns cost rates to states and costs to edges.

An edge $(\text{loc}, \psi, \text{act}, \lambda, \text{loc}') \in \text{Edg}$ defines a transition from location $\text{loc}$ to location $\text{loc}'$ taking an action $\text{act}$. This edge can only be traversed when the constraint $\psi$ over $\text{Cl}$ is true, and the set $\lambda \subseteq \text{Cl}$ identifies the subset of clocks which must be reset after the execution of the transition.

A trace of $P = (\text{Loc}, \text{loc}_0, \text{Cl}, \text{Act}, \text{Edg}, \text{Inv}, \text{Cost})$ is a sequence of locations and transitions $TR = \text{loc}_0 \xrightarrow{\text{act}_0} \text{loc}_1 \xrightarrow{\text{act}_1} \cdots \xrightarrow{\text{act}_n} \text{loc}_n$ where:

- For every $i$, there is a transition $T_i = (\text{loc}_i, \psi_i, \text{act}_i, \lambda_i, \text{loc}_{i+1}) \in E$;
- For every $i$, $c_i = C(T_i) + t_i \cdot C(l_i)$ is the cost incurred in the transition;
- The initial valuation $V_0 = 0$ which assigns 0 to every clock in $\text{Cl}$;
- After each transition, there is a new clock valuation $V_{i+1} = (V_i + t_i)\lambda_i = 0$ obtained by increasing every clock in $X_i$ by $t_i$ and re-initializing all clocks in $\lambda_i$ to 0;
- Each valuation $V_{i+t}$ for $t < t_i$ must satisfy the invariant $\text{Inv}(l_i)$;
- The valuation $V_{i+t_i}$ must satisfy $\psi_i$ for every $i$.

Let $\parallel$ be the parallel product operator over price timed automata. That is given a set of PTAs $\{P_1, P_2, \cdots, P_n\}$, $P_1 \parallel P_2 \parallel \cdots \parallel P_n$ will denote the corresponding parallel product obtained by synchronizing the transitions of the component PTAs via joint signals. The formal definition of this operator is given in (Bengtsson and Yi, 2004).
4 FROM ATTACK TREES TO PRICE TIMED AUTOMATA

In this section we explain how an attack tree is transformed into a network of price timed automata. The proposed transformation is borrowed from the work of (Kumar et al., 2015).

First in Figure 3 we draw the price timed automaton corresponding to a basic attack step. The proposed PTA has five nodes. The considered BAS is activated when the input-signal activate_BAS? is received from the corresponding parent-node PTA. In order to execute this input-signal the condition Ready2Start(Val) == 1 must hold. The clock variable Time is reset to zero as soon as the BAS is activated. The attributes of the attacker are updated through the transition labeled with Val := Modify(Val,Time). At the end of the execution of the BAS the PTA reaches either state Succeeded_BAS or Failed_BAS. In order to reach the state Succeeded_BAS the condition Able2Succeed(Val) == 1 must hold.

In Figure 4, we propose a PTA which corresponds to a parent node connected to two children via an

AND gate. This PTA is activated after receiving the input-signal activate_Prt?. After that an activation output-signal is sent to each child PTA. If a success signal is received from both children then the parent PTA moves to its success state.

Similarly Figure 5 is an illustration of the PTA corresponding to a parent node connected to two children via an OR gate. In this case receiving a success signal from one of the two children is enough to guar-
Figure 5: A priced timed automaton for an OR gate and a parent node having two children.

Figure 6: Priced timed automaton corresponding to the global goal of the attacker.

Finally Figure 6 gives a PTA which corresponds to the execution of the global goal of the attacker. The output signal activate_root? will be the first action to be executed by the network of obtained PTAs.

5 TEST GENERATION AND EXECUTION

Test generation consists in extracting abstract test cases from the obtained network of PTAs. For this purpose we may use UPPAAL CORA (Behrmann et al., 2005; Rasmussen et al., 2004) which is an extension of the platform UPPAAL. This extension is enriched with additional variables used for optimal reachability analysis.

As already mentioned the proposed framework in this work is based on the TTCN-3 standard (ETSI, 2015). For this purpose, we will take advantage from the work of (Lahami et al., 2016; Lahami et al., 2012a). Next we give a brief recall about the main constituents of the TTCN-3 reference architecture as illustrated in Figure 7:

- **Test Management (TM):** manages the whole test process by starting and stopping tests;
- **Test Logging (TL):** manages all log events;
- **TTCN-3 Executable (TE):** runs the compiled TTCN-3 code;
- **Component Handling (CH):** places parallel test components and guarantees communication between them;
- **Coding and Decoding (CD):** encodes and decodes received from and sent to the TE;
- **System Adapter (SA):** adjusts the communication with the application or system under test;
- **Platform Adapter (PA):** implements the set of external functions.

![Figure 7: TTCN-3 Architecture (Lahami et al., 2016).](image)

At this level we are interested in defining a set of rules for transforming abstract test cases into concrete TTCN-3 tests.

The adopted transformation algorithm may be inspired by the following works (Axel Rennoch and Schieferdecker, 2016; Hochberger and Liskowsky, 2006; Ebner, 2004). Table 1 gives some examples of the rules to use to derive TTCN-3 tests from abstract test cases. These rules are concisely explained below:
• R1: This rule generates a new TTCN-3 module for each abstract test suite;
• R2: This rule transforms each test sequence into a TTCN-3 test case;
• R3: This rule associates a TTCN-3 timer with each abstract timed behavior;
• R4: This rule transforms each test sequence into a TTCN-3 function;
• R5: This rule transforms the abstract channels into TTCN-3 templates.

Table 1: TTCN-3 Transformation Rules.

<table>
<thead>
<tr>
<th>R#</th>
<th>Abstract Concepts</th>
<th>TTCN-3 Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Test Suite</td>
<td>TTCN-3 Module</td>
</tr>
<tr>
<td>R2</td>
<td>Single Trace</td>
<td>TTCN-3 Test Case</td>
</tr>
<tr>
<td>R3</td>
<td>Timed Behavior</td>
<td>TTCN-3 Timer</td>
</tr>
<tr>
<td>R4</td>
<td>Test Sequence</td>
<td>TTCN-3 Function</td>
</tr>
<tr>
<td>R5</td>
<td>Channel</td>
<td>TTCN-3 Template</td>
</tr>
</tbody>
</table>

Cloud computing can be used in the field of software testing to deal with the problem of lack of resources and the considerable cost of building a distributed test solution during the testing activity. Consequently, the notion of Cloud testing is increasingly emerging in order to offer cost-effective and efficient testing facilities. As defined by (Gao et al., 2011), it corresponds to testing activities (namely test case generation, test case execution and test result evaluation) on a cloud-oriented environment.

The proposed cloud testing architecture is built based on TaaS (Testing as a Service) concepts. Figure 8 outlines an overview of its different components of this architecture.

• Test Management GUI: offers a GUI (Graphical User Interface) charged with manages the whole testing process.
• Resource Management: enables flexibility and elasticity during the testing process.
• Test Component Management: offers services which create/delete test components and start/stop their execution.
• Runtime Monitoring: gives the status of the resources of each VM (e.g., memory, CPU, etc.).

6 RELATED WORK

Authors of (Felderer et al., 2016) proposed an interesting survey on dozens of articles related to model-based security testing chosen from the most relevant digital sources and classified with respect to specific criteria. However this review did not cover any work dealing security issues for IoT and smart cities. In the opposite way, the authors of (Ahmad et al., 2016) presented a model-based approach to test IoT systems but they did not consider security aspects in anyway. Moreover the authors of (Wang et al., 2017) proposed a formal framework based on timed automata for analyzing security properties of cyber-physical systems. In (Krichen et al., 2018a), the authors proposed a preliminary work which introduced a model based approach for testing security aspects of IoT systems in smart cities. Regarding the use of attack trees we mention the following works (Aslanyan et al., 2016) (Kammüller et al., 2016) (Kumar et al., 2015) (Kordy et al., 2014) which adopted this formalism to model and analyse security attacks. However none of these works has attempted to use testing techniques to check the ability of the considered systems to defend themselves against security attacks.

7 CONCLUSION

In this work we proposed a new approach for testing security aspects for IoT systems in Smart Cities. The proposed approach is based on the use of attack trees which correspond to a graphical representation of the strategy adopted by an attacker in order to violate the IoT system. We proposed a transformation method to translate a given attack tree into a network of price timed automata. The latter is then used as input for the test generation algorithm for producing abstract test cases. The obtained test cases are translated into concrete TTCN-3 test scenarios. Finally a cloud oriented testing architecture is proposed in order to execute tests and collect testing results.

Many extensions are possible for this work. First we may attack advantage from the work of (Lahami et al., 2016; Krichen, 2012; Lahami et al., 2012b; Krichen and Tripakis, 2006) to build a decentralized testing architecture. Moreover we may adopt the methodology proposed in (Krichen et al., 2018b; Maâlej and Krichen, 2016; Maâlej et al., 2013; Maâlej et al., 2012b; Maâlej et al., 2012a) to combine security and load tests for IoT applications. Finally we may exploit the same techniques presented in (Ben-salem et al., 2007) in order to refine abstract test cases before translating them into TTCN-3.
REFERENCES


