Automatically Verifying STRAC Policy

GUO Yunchuan[1,2], YIN Lihua[1,3*], LI Chao[1]
[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2]Guangxi Key Laboratory of Trusted Software (Guilin University of Electronic Technology)
[3]Beijing Key Laboratory of IOT Information Security (Institute of Information Engineering, CAS)
{guoyunchuan, yinlihua}@nelmail.iie.ac.cn

Abstract—In the IoT (Internet of Things), inconsistent ACP (Access Control Policies) would cause disastrous consequences, for example, fire disasters and failures of cardiac pacemakers, as a result, ACP should be verified before they are applied. Tediousness and error-proneness of manual verifications make automatic and formal verification necessary. In this paper, timed automata is presented to formally model the STRAC (Spatio-Temporal Access Control based on Reputation, one policy for the IoT) policies and CTL (Computation Tree Logic) is adopted to describe the properties that should be satisfied by these policies. Then model checker UPPAAL is proposed to automatically verify whether STRAC policies conform to security properties. Experiment results show that our approach is effective.

***Index Terms*—Access control, security policy, formal methods**

I. INTRODUCTION

As a dynamic global ubiquitous network, the Internet of Things (IoT) links physical and virtual objects by integrating sensors and smart terminals and will create hundreds of billions of dollars in savings and productivity gains for businesses, governments and households. Obviously, the lack of security will severely hinder the development of the IoT. Access control, determining "who is authorized to access specific objects and in what mode (e.g., read, write)", is central component of the IoT security[1].

Motivation. The following critical characteristics of the IoT make it difficult for designers to design consistent ACP (ie, no conflicting access decisions occur) to provide security: (1) Cross-domain nature. A number of terminal nodes distributed in wide areas are owned by multiple organizations to deal with their respective tasks, some of which are even conflict. (2) Complexity. A strict access control policy can contain tens of thousands of rules. In practice, these complex polices are often manually formulated, as a result, many errors and inconsistencies might be raised because of error-proneness of manual formulation. These errors and inconsistency can put system resources at risk unnecessarily and bring be unpredictable and even disastrous consequences, for example, fire disasters and failures of cardiac pacemakers. As a result, ACP should be verified before they are applied. Tediousness and error-proneness of manual verifications make automatic and formal verification necessary.

Contributions. In this paper, timed automatons are used to formally model the STRAC (Spatio-Temporal Access Control based on Reputation) policies and CTL (Computation Tree Logic) is adopted to describe the properties that should be satisfied by STRAC policies. Then model checker UPPAAL[2] is proposed to automatically verify whether the IoT policies conform to STRAC properties. Experiment results show that our approach is effective.

II. AUTOMATA MODEL

Generally, access control for the IoT includes three components: controlling terminals, a reference monitor and controlled Devices (simplified as *Term*, *RM,* and *Device* respectively in the sequels). In this scenario, *Terms* send access requests to the *RM*. Upon receiving these requests, *RM* retrieves the stored access rule table, decides whether to approve these requests, and sends the result to the controlled *devices*.

To automatically verify ACP, we have to formally model the behaviors of these three components. In the IoT, it is necessary to describe operations that can only be executed within a given time period, as a result, we choose a timed automaton, which extends a finite automaton with a finite set of real-valued clocks, as the tool to model access control components. For simplicity, we take smart home applications as an example to illustrate how access control components are formally modeled.

Our example includes four smart devices (a TV set (*TV*), an air conditioning (*AC*) unit, a microwave (*MW*), and an electric rice cooker (*RC*)), which can be remotely or locally controlled by parents and their children in the office, school, and home. We stipulate that: (1) both *TV* and *AC* can be closed or opened remotely or locally; (2) *RC* can only be opened locally but can be closed remotely; (3) *MW* can be configured and opened by parents either remotely or locally but cannot be opened by children remotely; (4) Because of the limitations regarding power load, the maximum number of devices than can able to run simultaneously is set to three; (5) Every device only runs during the specified time.

Formally, we can use STRAC to summarize the above rules, as follows. ID={0,1,2} is a set of terminal IDs based on the assumptions that (1) Terminal 0 is owned by children and is used in school and at home, (2) Terminals 1 and 2, owned by parents are used at school, office, and home, and (3) REP={*highRep*, *lowRep*}, where *highRep* and *lowRep* denote high reputation and low reputation, respectively. We assume that the reputation of Terminal 0 is low and the others are high. LT={*TVtime*, *ACtime*, *MWtime*, *RCtime*}, where each element respectively represents time

*YIN Lihua is the corresponding author.

when *TV*, *AC*, *MW,* and *RC* can run. Location *LOC*={*home*, *office*, *school*}, Operation *OP*={*open, close*, *config*}, Object *O*={*TV*, *AC*, *MW*, *RC*}, Permission *PERM*={*open*, *close*}×*O*∪{(*config*,*MW*)},

An access request consists of six parameters: *tm_rm_id* (ID of the *term*), *tm_rm_loc* (location of the *term*), tm_rm_time (current time), *tm_rm_dev* (target *device*), *tm_rm_op* (operation on the *device*), and *tm_rm_MW_time* (timer for *MW*). Fig. 1 gives the timed automata of terminals, where *id* is an input parameter, *TermToRM[id]* is a channel, *loc_t* is a set of all locations(*home*, *school,* and *office*). The remaining variables are similar to *loc_t*.

As shown in Fig. 2, *RM* receives access requests from the channel *TermToRM*[*id*], checks the state (busy or free) of the controlled device, and makes a decision according to the reputation and location of the terminals and the current time. This decision is then sent to the corresponding device.

Fig. 3 illustrates the timed automata of controlled devices. The initial state of each device is DevOff. When the instruction *open* arrives, the corresponding device starts to run. If the instruction *close* arrives, the corresponding device stops running. In *MW*, *dev_config* is used to store the time parameter of the *MW* timer.

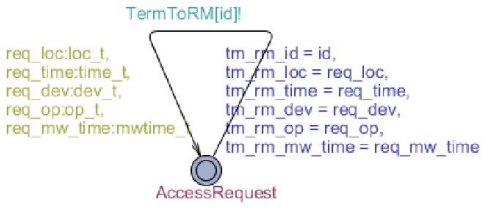

Fig. 1. Timed automata of *terminals*

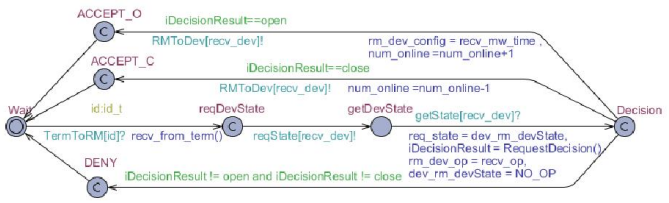

Fig. 2. Timed automata of RM

The composition (*Terminals* || *RM* || *Device*) of the above three timed automata forms a timed automation network, and we can use this network to verify STRAC policies.

III. Policies and their Verification

In our studies, three properties are verified:

(1) "Deadlock must be avoided in smart home applications". This can be specified as "A[] not deadlock" by using CTL;

(2) The number of simultaneously running devices is always less than or equal to three. This can be specified as "A[] not (Device(TV).DevOn and Device(AC).DevOn and Device(RC).Heating and Device(*MW*).DevOn)";

(3) When the parents open a microwave in office, the microwave would always switch to the *Ripe* state, which is "(tm_rm_id==1&&tm_rm_loc==office&& tm_rm_time ==8&&tm_rm_dev==*MW*&& tm_rm_op== open) -->Device(*MW*).Ripe", where the reputation of Terminal 1 (belonging to the parents) is high and the current time (m_rm_time==8) is within the work time of *MW*.

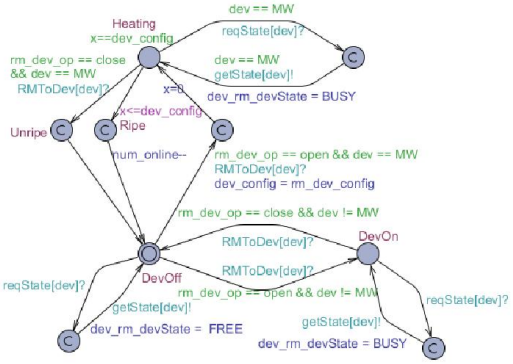

Fig. 3. Timed automata of controlled devices

We translated the above three time automata into the UPPAAL language to verify the three properties, and the result shows that the first two properties are true. This means that the system is deadlock-free and that the number of simultaneously online devices is always less than or equal to three. However, property (3) is false, meaning that, even when parents start the microwave while in the office, the microwave does not necessarily switch to the *Ripe* state. The corresponding counter-example given by UPPAALL is as follows: If parents start the microwave and the rice cooker at the same time in the office, the child will not be able to start the air-conditioning and watch *TV* due to the power load limitation. In this case, children could stop the microwave, thus preventing it from switching to the *Ripe* state. This example demonstrates that the STRAC policies can be checked by UPPAAL.

IV. Conclusion

As one of the key technologies involved in providing security, access control – determining *who* is allowed access, *when* access is permitted, and *where* access takes place is central component of the IoT security. Policy verification is important in the access control. In this paper, model checker UPPAAL is proposed to automatically verify whether the IoT policies conform to security policies. Experiment results show that our approach is effective. In the future, we develop the more efficient approach for verifying the policy of the IoT.

Acknowledgements:

This work was supported by the National High Technology Research and Development Program of China (2013AA014002) and the National Natural Science Foundation of China (61100181, 61100186, 61262008, 61363030).

References

1. Gusmeroli, S., S. Piccione, and D. Rotondi, *A capability-based security approach to manage access control in the Internet of Things.* Mathematical and Computer Modelling, 2013. **58**(5): 1189-1205.

2. Gómez, R., *Model-checking timed automata with deadlines with UPPAAL.* Formal Aspects of Computing, 2013. **25**(2): 289-318.