Automatic Trigger Generation for End User Written Rules for Home Automation

Chandrakana Nandi
Department of Computer Science and Engineering
University of Washington, Seattle
cnandi@cs.washington.edu

ABSTRACT

To customize the behavior of a smart home, an end user writes rules. When an external event triggers the rule, it executes; for example, when the temperature is above a certain threshold, then window awnings might be extended. End users often write incorrect rules [6]. This paper's technique prevents a certain type of mistake: *errors due to too few triggers*. It statically analyzes a rule's actions to determine what triggers are necessary. We have implemented the technique as a tool named TrigGen and tested it on 116 end-user written rules for an open-source home automation platform called openHAB. TrigGen identified that 66% of the rules had fewer triggers than required for correct behavior. The missing triggers could lead to unexpected behavior and security vulnerabilities in a smart home.

1. RESEARCH PROBLEM

Most home automation platforms support end-user customization components: a user writes rules to determine what actions should be taken by what device under what conditions [10]. A recent study [13] has shown that rule-based systems are one of the most practical solutions for enabling end-users to customize the behavior of their smart homes. A rule-based language has two main components: triggers that cause a rule to be fired and actions to be executed when a rules fires.

```
rule "Away rule"
when
  // trigger block
  Item State_Away changed
then
  // action block
  if(State_Away.state == ON){
    if(State_Sleeping.state != OFF){
      postUpdate(State_Sleeping,OFF)
    }
  }
end
```

Listing 1: End user written rule for setting the Away or Sleeping state in the house.

Listing 1 shows an example of a rule. The part between `when` and `then` is the trigger block and the part between `then` and `end` is the action block. This programming paradigm is also called Trigger Action Programming (TAP). Even though TAP is the most commonly used and practical approach for home automation, end users often make errors in writing trigger-action programs [6],[14]. These errors could be in 1) writing the triggers, or 2) writing the actions or 3) both. In a heterogeneous system like a smart home where there are multiple interacting devices, it is common for the rules to interact. As a result, an error in one rule can propagate to others and cause unexpected behavior or security vulnerabilities in different parts of the house. In this paper, we propose a solution to make it easier for the end users to write home automation rules correctly. Our approach eliminates one category of errors in the rules—*errors due to too few triggers*. By doing a static analysis of the actions, our approach automatically generates the trigger conditions so that the user does not have to worry about including the *correct* and *sufficient number* of triggers in the rules.

Consider the rule which is written by an end user in listing 1. This rule is supposed to set the value of `State_Away` to `ON` when no one is in the house. It also sets `State_Sleeping` to `OFF` if it is currently `ON` so that both `State_Away` and `State_Sleeping` are not `ON` simultaneously. However, the trigger for this rule only contains `State_Away`. As a result, this rule is only fired when `State_Away` changes. If the value of `State_Sleeping` changes after the value of `State_Away`, then the rule is not fired due to which, it allows both values to be set at the same time. This deviates from the expected behavior of the rule—it is no longer clear whether the home inhabitants are away or inside sleeping. This example shows that even if the action block of a rule is implemented correctly, not having all the correct triggers can lead to too few firings of the rule and thus, unexpected behavior. This paper's technique prevents this problem. We have implemented it as a tool named TrigGen and used it to analyse 116 home automation rules written by end users—TrigGen found 66% rules to be deviating from the expected behavior due to lack of trigger conditions.

2. RELATED WORK

Security loopholes in smart homes have been investigated in several papers [3, 4, 12, 9]. Recently, Fernandes et al.[5] did a security analysis of several apps based on Samsung SmartThings and discovered that many of them unnecessarily granted full access to the devices in the house. While they aimed at identifying security flaws in the SmartThings framework itself, our aim is to assist end users in writing correct automation rules and we do so by automatically generating the trigger conditions. Further, instead of analysing apps for home automation, we analysed end user written rules which are mostly written by non-programmers. Some

work has been done on detecting conflicts in trigger-action programs [7, 8, 11]. In addition to detecting conflicts in triggers, TrigGen is capable of automatically generating the correct triggers for a rule which makes it unique and different from previous work.

3. APPROACH

We have developed a static technique to automatically generate correct event-based triggers from the actions written by the end users. Our tool, TrigGen has three features—1) generating all event-based triggers, 2) detecting missing triggers when some of them are already written by the end user, and 3) detecting conflicts by identifying all rules which try to write to the same item. Figure 1 shows the design of our tool. It takes as input a *rule* file which contains all the end user written rules and an *item* file which contains a list of all the smart devices (or items) installed in the home.

The rule file is parsed to extract the action block and the item file is parsed to extract all the item names, types (`Switches`, `Shutters`, `Dimmers` etc.), and the groups to which they belong; for example all the light switches in the living room may be grouped together.

By statically analysing the abstract syntax tree of the action block A of rule R, TrigGen first identifies all the items that appear in R. This is an exhaustive list of all potential event-based triggers. Let this list be T. Naively adding all $t \in T$ as triggers of R would either unnecessarily fire R too many time or add wrong triggers. Hence, we apply an elimination technique to get rid of triggers that are not required or are wrong. For that, we define the following terms.

Definition 1. An item is *live* in R if its value is read before it is written to in the action block, A.

Definition 2. An item is *dependent* in R if its value is computed based on other items or local variables in the rule—this implies that the item is *not live*.

Definition 3. An item is *independent* in R if its value is directly obtained from a sensor or a user's input and not computed based on any other item or variable—this implies that the item is *live* in R.

Definition 4. An event-based trigger is *redundant* if inclusion of the trigger in a rule *never* changes the state of any item or the value of any variable involved in A when the rule is fired due to it.

Trigger elimination strategy. **An item d which is not *live* in R is not included in T because it is *redundant*. As a corollary, if d is *dependent*, it is also not included in T.**

Conflicting rules. Two or more rules are said to have a conflict for an item if they all write to the item. TrigGen can detect such conflicts between rules and warn the end user about potential conflicts.

4. EXPERIMENTAL RESULTS

We evaluated our tool on 116 end user written rules. We obtained the rules from links provided on the openHAB website [1],[2]. It took $\sim$ 7.12 ms for the tool to generate the triggers and find the missing triggers for all the rules. For 109 rules, our tool suggested a list of all required event-based triggers and for 77 rules, our tool found missing triggers when some triggers were already provide by the users. The maximum number of triggers TrigGen found for a rule is 27. Figure 2 shows the output of TrigGen for suggesting triggers.

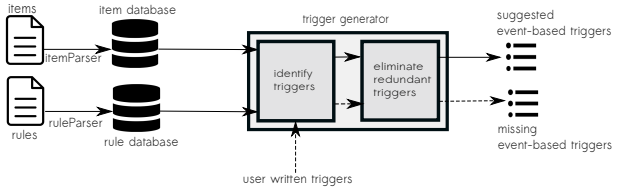

Figure 1: Design of TrigGen: ruleParser and itemParser parse the rule and item files respectively to extract useful information. This information goes to the core trigger generator component which enumerates all possible triggers and eliminates redundant ones. If some of the triggers are already written by the end user, TrigGen can detect the missing ones.

For 18 rules, our tool detected potential conflicts among one or more rules. The maximum number of conflicting rules TrigGen found for an item is 2.

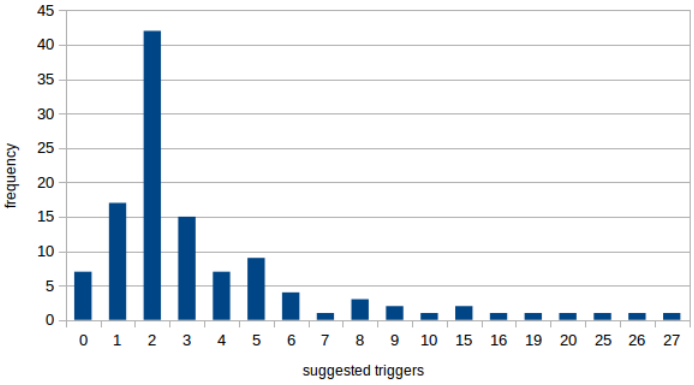

Figure 2: Summary of the output of TrigGen for suggesting triggers. The x-axis shows the number of triggers and the y-axis shows the number of rules.

5. CONCLUSIONS

Not surprisingly, our research shows that end user written rules are often incorrect. We observed that a common error made by end users while writing the rules is having insufficient number of triggers, leading to fewer firings of the rules than necessary. To prevent this problem, we developed TrigGen that can automatically generate trigger conditions based on the actions in a rule, identify missing triggers if some of them are already provided by the end user and also identify conflicting rules. This reduces the burden of the end users by assisting them in including all the required triggers and also helps them detect errors in the rules.

Scope. Some rules are meant to be fired under temporal triggers only (such as at a certain time of the day). While our tool is capable of generating all event-based triggers, it cannot generate such temporal triggers. This is because it runs directly on the rules without any additional annotations from the end users about their preferences and without that, it is impossible to infer the exact temporal triggers the user expects.

6. REFERENCES

[1] Configs, Tools & Icons. http://www.intranet-of-things.com/software/downloads. Accessed: May 2016.

[2] openhab downloads. http://www.openhab.org/getting-started/downloads.html. Accessed: May 2016.

[3] T. Denning, T. Kohno, and H. M. Levy. Computer security and the modern home. *Commun. ACM*, 56(1):94–103, Jan. 2013.

[4] N. Dhanjani. Abusing the internet of things: Blackouts, freakouts and stakeouts.

[5] E. Fernandes, J. Jung, and A. Prakash. Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*, May 2016.

[6] J. Huang and M. Cakmak. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, pages 215–225, New York, NY, USA, 2015. ACM.

[7] H. Luo, R. Wang, and X. Li. A rule verification and resolution framework in smart building system. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 438–439, Dec 2013.

[8] C. Maternaghan and K. J. Turner. Policy conflicts in home automation. volume 57, pages 2429–2441, New York, NY, USA, Aug. 2013. Elsevier North-Holland, Inc.

[9] S. Mennicken, J. Vermeulen, and E. M. Huang. From today's augmented houses to tomorrow's smart homes: New directions for home automation research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 105–115, New York, NY, USA, 2014. ACM.

[10] M. W. Newman. Now we're cooking: Recipes for end-user service composition in the digital home, 2006. Position Paper–CHI 2006 Workshop IT@Home.

[11] Y. L. Sun, X. Wang, H. Luo, and X. Li. Conflict detection scheme based on formal rule model for smart building systems. volume 45, pages 215–227, 2015.

[12] B. Ur, J. Jung, and S. Schechter. The current state of access control for smart devices in homes. In *Workshop on Home Usable Privacy and Security (HUPS)*. HUPS 2014, July 2013.

[13] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 803–812, New York, NY, USA, 2014. ACM.

[14] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3227–3231, New York, NY, USA, 2016. ACM.