Research Article

TapChain: A Rule Chain Recognition Model Based on Multiple Features

Keyu Jiang,¹ Hanyi Zhang,² Weiting Zhang,¹ Liming Fang^{1,3,4}, Chunpeng Ge,^{1,3,4} Yuan Yuan,² and Zhe Liu¹

¹College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211 100, Jiangsu, China

²Alibaba Group, Hangzhou 311 100, Zhejiang, China

³Nanjing University of Aeronautics and Astronautics Shenzhen Research Institute, Shenzhen 518 000, Guangdong, China

⁴Science and Technology on Parallel and Distributed Processing Laboratory (PDL), Changsha 410 000, Hunan, China

Correspondence should be addressed to Liming Fang; fangliming@nuaa.edu.cn

Received 24 September 2021; Revised 27 October 2021; Accepted 18 November 2021; Published 23 December 2021

Academic Editor: Jinguang Han

Copyright © 2021 Keyu Jiang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Trigger-action programming (TAP) is an intelligent tool, which makes it easy for users to make intelligent rules for IoT devices and applications. Unfortunately, with the popularization of TAP and more and more rules, the rule chain from multiple rules appears gradually and brings more and more threats. Previous work pays more attention to the construction of the security model, but few people focus on how to accurately identify the rule chain from multiple rules. Inaccurate identification of rule chains will lead to the omission of rule chains with threats. This paper proposes a rule chain recognition model based on multiple features, TapChain, which can more accurately identify the rule chain without source code. We design a correction algorithm for TapChain to help us get the correct NLP analysis results. We extract 12 features from 5 aspects of the rules to make the recognition of the rule chain more accurate. According to the evaluation, compared with the previous work, the accuracy rate of TapChain is increased by 3.1%, the recall rate is increased by 1.4%, and the precision rate can reach 88.2%. More accurate identification of the rule chain can help to better implement the security policies and better balance security and availability. What's more, according to the rule chain that TapChain can recognize, there is a new kind of rule chain with threats. We give the relevant case studies in the evaluation.

1. Introduction

Trigger-action programming (TAP) [1] is a programming method that users can use to stitch devices and applications (APPs) together, such as IoT devices, Twitter, and Google Calendar. The users do not need to know the specific programming language. They just need to use the graphical interface to create rules such as “if this, then that.” For example, the rule “if your room temperature is too high, then turn on your A/C” indicates that the air conditioner will be turned on when the temperature in your room is too high. TAP can be used not only in Internet of Things but also in the interaction between APPs. For example, the rule “if you share a photo on Instagram, then upload a photo from URL in your Facebook” instructs the photo will be uploaded in

your Facebook when you share a photo on Instagram. This kind of personalized and simple programming model provides great convenience for end-users.

Unfortunately, as more and more devices and APPs need to be associated, this intelligent method will also bring vulnerabilities to the end-user system due to the formation of a variety of rule chains vulnerabilities. For example, one rule turns on the heater, while another rule opens the window when the temperature is too high. These two rules may cause windows to open unexpectedly and then the open windows can be exploited by attackers. However, a rule adds any new iOS reminders to your Google Calendar, and another rule posts a tweet when any new event is added to your Google Calendar. These two rules may upload reminders you do not want to upload to public space to Twitter, as shown in Figure 1.

FIGURE 1: Application scenario of TAP.

In previous work, researchers pay more attention to the construction of a security model to identify threats such as SOTERIA [2] but ignored the identification of the rule chain. Although the construction of the security model is important, the accurate identification of the rule chain is also crucial. Only by accurately identifying the rule chain, the excellent security model can work more perfectly. Celik et al. [3] can accurately identify the rule chain in SmartThings by using code analysis tools on the SmartThings platform. However, it is difficult to apply the closed nature of IoT platforms, such as IFTTT, on a large scale [3]. Although Wang et al. [4] can identify the rule chain without source code, its accuracy and precision are not good enough.

In this paper, we propose a rule chain recognition model based on multiple features, TapChain. We define a $\langle \text{action}, \text{trigger} \rangle$ pair as a rule chain. TapChain can improve the recognition accuracy of the rule chain through the natural language description of rules. To overcome the closed nature, we use natural language processing (NLP) technology to analyze and process trigger description and action description, respectively. Due to the marking error of the NLP tool, we design a correction algorithm to help us get the correct mark. Since there is no label data of rule chain at present, we manually combine and identify $\langle \text{action}, \text{trigger} \rangle$ pair in the same service and label 0 or 1 for each pair to indicate whether the rule chain will be formed. Then, we extract 12 features from trigger description and action description and then judge whether an action will cause a trigger to start.

We evaluate TapChain according to triggers and actions from 279,828 real rules on the IFTTT website. We find that TapChain can achieve 89.9% accuracy, 88.2% precision, and 92.1% recall in identifying rule chains. Compared with the previous work, TapChain's recognition accuracy of the rule chain is improved by 3.1% and the recall rate is improved by 1.4%. In addition, through TapChain, we find a new kind of rule chain with threat and we give a case study.

We summarize our contributions as follows:

- (1) We propose TapChain, a rule chain recognition model based on multiple features. We extract 12 features to identify the rule chain. These features can

more finely represent the relationship between action and trigger in $\langle \text{action}, \text{trigger} \rangle$.

- (2) We design a correction algorithm for TapChain to help us get the more accurate NLP analysis results.
- (3) Compared with the previous work, TapChain's recognition accuracy of the rule chain is improved by 3.1%, the recall rate is improved by 1.4%, and the precision rate can achieve 88.2%.
- (4) Thanks to the improvement of the accuracy of rule chain recognition, we find a new kind of rule chain with threats. We give the relevant case study in the evaluation.

The organization of this paper is as follows. Section 2 outlines the related work. Section 3 shows the data processing. Section 4 introduces TapChain. In Section 5, we evaluate TapChain. Section 6 discusses usability and limitations. Section 7 concludes the paper.

2. Related Work

Comprehension of Single Rule in TAP. Researchers have been studying TAP for seven years. The earliest research can be traced back to 2014. To test the usability of TAP, Ur et al. [1] gathered 266 participants to carry out a user study, which laid a good foundation for the research of TAP. Since then, many researchers [5–9] have been devoted to the research in this field. In 2020, Zhao et al. [10] proposed a visual interface to interpret TAP rules to help users understand the operation of rules. These researches are of great significance for other researchers to assume that users can understand a single rule when they study the rule chain.

Security of Single Rule in TAP. In 2016, Fernandes et al. [11] researched the potential security problems caused by TAP on the SmartThings platform for the first time and found the vulnerabilities in the event subsystem of SmartThings through static analysis. Then, some researches [12–16] summarized the security problems of a single rule and put forward corresponding solutions. The security of a single rule is also helpful to the security of the rule chain.

Security of Rule Chain in TAP. On the SmartThings platform, Celik et al. [2, 3] identified the threats in TAP and proposed corresponding solutions. However, their solution needs to obtain source code, which is difficult for the IFTTT platform, because the IFTTT platform is a closed source platform. Wang et al. [4] proposed iRule, which not only constructs the security model but also realizes the identification of the rule chain without obtaining the source code. However, the performance of iRule in the chain of recognition rules can be optimized. They only extracted the features of the predicate and object from the description. It is difficult to make an accurate judgment when identifying the rule chain which has a great difference between predicate and object, especially when it involves environmental factors. Our work is to improve the recognition rule chain to better identify more threat rule chains.

3. Data Processing

In this section, we describe how we process data. Then, we introduce how we label the rule chain.

3.1. Dataset. Our work is based on the rules on the IFTTT website. So, first we need to obtain the data. We try to use Blase et al.'s method [6] to obtain the latest rules. However, at present, the links to the rules on the IFTTT website have been processed additionally. Although the URL of each rule is still "https://ifttt.com/applets/ID," the ID of each rule is no longer a number, but a combination of numbers and letters, such as "https://ifttt.com/applets/SdbAq2ce." At present, it is difficult for us to find crack methods from the ID of rules. Therefore, we have to use the dataset provided by Mi [17]. The dataset contains rules, triggers, and actions.

3.2. Data Preprocessing. It seems that we obtain a lot of rules, but there is little practical help for us to identify the rule chain. This is because each rule is composed of a trigger and an action. When users make rules, they can combine trigger and action in a variety of ways. But in the end, trigger and action come from the trigger dataset and action dataset, respectively. Therefore, we can decompose the rules into trigger and action, analyze them respectively, and then find the rule chain. Besides, the dataset contains a lot of invalid information, so we need to clean the data. For example, in the triggers dataset, "triggerUrl," "triggerchannelURL," and so on, are invalid information. So, we need to filter out this information.

3.2.1. Trigger Dataset. The trigger dataset is used to determine the detail of each trigger for comparison with action. We filter the trigger to keep only the information we need. There is the following example:

- (1) **triggerTitle:** door opened
- (2) **triggerDesc:** this trigger fires when the selected door is opened
- (3) **triggerChannelName:** abode
- (4) **triggerFieldList:** ["Select door"]

Among them, "triggerTitle" and "triggerChannelName" can uniquely determine a trigger. "TriggerDesc" describes the detail of the trigger. "TriggerFieldList" is supplementary to "triggerDesc" and it is a personalized value that can be set by users. In this example, we can use the "triggerFieldList" to determine which door the trigger exactly corresponds to. "TriggerFieldList" is also extracted as features.

3.2.2. Action Dataset. The action dataset is used to determine the function and impact of each action for comparison with a trigger. We also sift out the information we need. There is the following example:

- (1) **actionTitle:** change mode
- (2) **actionDesc:** this action will change your abode system to the selected system mode
- (3) **actionChannelName:** abode
- (4) **actionFieldList:** ["Which mode?"]

"ActionTitle" and "actionChannelName" can uniquely determine an action. "ActionDesc" describes the function and influence of the action in detail. "ActionFieldList" is supplementary to "actionDesc" and it is a personalized value which can be set by users. In this example, we can use the "actionFieldList" to determine which mode the action will change to.

3.3. Description Processing. First of all, at the beginning of the description of each trigger, there are expressions such as "this trigger fires when." Similarly, at the beginning of the description of each action, there are expressions such as "this action will." This kind of information is not only helpful for us to analyze the rule chain but also increases the analysis complexity of the description statement. So, we delete these prefixes and capitalize the first letter of the new first word to make it a sentence.

Second, we find that some trigger descriptions have multiple sentences. For example, the description "this trigger fires every time anyone shares a public photo with a tag you specify. Note: limited to 30 photos per check" contains two sentences. We notice that the first sentence of the description mainly describes the function of the trigger. The rest of the sentence is usually a prompt to the user. So, we only keep the first sentence of the description.

Third, in the description, some special symbols appear, such as % and /. For %, it often appears in statements describing percentage, such as humidity level and lighting level. We convert it into a word, percentage, to avoid errors in NLP tool analysis. For /, it often appears in the abbreviation of air conditioning. We find that A/C often appears in the description instead of air conditioning. We directly delete these /, to avoid the NLP tool taking A/C apart for analysis.

3.4. Dataset Labeling. When we look for the rule chain, we must find which trigger can be affected by the action. We define the pair as <action, trigger>. If an action can start a trigger in <action, trigger>, we label it as 1; otherwise, we label it as 0.

For the convenience of describing our method, we give some examples of action and trigger, as shown in Table 1. ID is used to distinguish each data represents action or trigger, A represents action, and T represents trigger. The description is the description of the action or trigger.

We divide $\langle \text{action}, \text{trigger} \rangle$ labeled as 1 into two parts. One is that action can start trigger directly, which is called direct chain, and the other is that action can start trigger by affecting the physical environment, which is called indirect chain. For the two types of data, we describe them separately.

3.4.1. Direct Chain. We define that action can only start triggers belonging to the same services. When we label, comparing the description of action and trigger, we can judge whether most $\langle \text{action}, \text{trigger} \rangle$ can form a direct chain. For example, A1 and T1 in Table 1; we can directly judge that A1 can start T1. First of all, A1 and T1 belong to the same service. Secondly, according to the description of T1, as long as the mode of abode system changes, it can be triggered. A1 is to change the mode of abode system. Finally, we need to determine whether the values in “FieldList” are the same. Therefore, we can label $\langle A1, T1 \rangle$ as 1 if the mode of action is the same as that of the trigger; otherwise, label 0.

3.4.2. Indirect Chain. In contrast, labeling the indirect chain is more difficult. We first need to define the environment for the device. For example, the operation of air conditioning and heaters can affect both temperature and humidity. We have made a summary of the devices appearing in the triggers dataset and actions dataset, as shown in Table 2.

The second difficulty is that we need to distinguish how devices affect the environment. For example, air conditioning has many functions, such as refrigeration and heating. Refrigeration will reduce the temperature and humidity, but heating will increase the temperature and reduce the humidity. “FieldList” can help us.

For example, A2 describes that this action will turn on your A/C in the specified room and run it in temperature mode. From the description, it is difficult to determine the impact of air conditioning on the environment. However, the “keep temperature at” in “actionFieldList” clearly indicates that the function of A2 is to keep the temperature at a certain value. Therefore, we can know whether A2 can start T3 by comparing the “temperature threshold” in A2 “actionFieldList” with the “keep temperature at” in T3 “triggerFieldList.” If the value in the “actionFieldList” of A2 is less than the value of the “triggerFieldList” of T3, we can label $\langle A2, T3 \rangle$ as 1; otherwise, we label 0. At the same time, since A2 turns on the air conditioner, T2 may be started. If the first value in T2 “triggerFieldList” is “turn on” and the second value is the same as the first value in A2 “actionFieldList,” we can label $\langle A2, T2 \rangle$ as 1; otherwise, we label 0.

Based on the above analysis, the personalized values, values in the “actionFieldList” and “triggerFieldList” are very important for us to judge whether a rule chain can be formed. So, we randomly set these values to complete labeling.

4. TapChain

In this section, we describe TapChain, our model to recognize the rule chain. Because the IFTTT platform is a closed source platform, program analysis technology is difficult to apply to it [4]. To overcome this difficulty, TapChain recognizes the rule chain by using natural language processing (NLP).

4.1. Overview. TapChain first takes the data processed in Section 3 as the input. These data are divided into two categories. One is “Desc” data, including “triggerDesc” and “actionDesc.” The other is “FieldList” data, including “triggerFieldList” and “actionFieldList.” For “desc,” we use the NLP tool, Stanford CoreNLP [18], to perform part-of-speech (POS) tagging and dependency parsing on sentences. When the tool is used for sentence analysis, some special sentences are analyzed inaccurately, so we correct these inaccurate analysis results. Then, we extract semantic related features from the corrected results. For “FieldList,” the data inside are personalized, and users can set it by themselves. To overcome the uncertainty brought by personalized data, we abstract the data into the comparison of the same value or the same range. Then, we extract the “FieldList” features. We also extract environment-related features based on “FieldList” and “Desc.” Finally, we input the extracted features into the machine learning models to obtain the recognition results of the rule chain. The overview is shown in Figure 2.

4.2. Description Analysis Based on NLP. We first perform part-of-speech (POS) tagging and dependency parsing using Stanford CoreNLP. The generated dependency tree is shown in Figure 3.

The dependency relationship analyzed by Stanford CoreNLP is complex. There are some invalid elements, such as the analysis of stop words and punctuation. So, we summarize the syntax elements that we focus on, as shown in Table 3.

- (1) Predicate: it is a verb, expressed as root. Sometimes, there is a verb phrase. At this time, we need to combine the word marked as root with the word marked as compound:prt as verb, such as “turn on.” When conjunctions appear in a sentence, there will be coordinate verbs, expressed as conj.
- (2) Direct object: the action is executed on it. The direct object is expressed as obj. When there is no word marked obj in the sentence, we take the word marked obl as the direct object, because indirect objects cannot exist independently. We also define the subject in the passive voice as the direct object, expressed as nsubj:pass.
- (3) Indirect object: the indirect object is expressed as obl, such as “mode” in “change your abode system to the selected system mode.”
- (4) Modifier: modify a noun. In our work, most of them modify the object, such as “new” in “write a new message in a given sphere.”

TABLE 1: Example of action and trigger.

ID	Service	Description	FieldList
A1	Abode	Change your abode system to the selected system mode	["Which mode"]
A2	Ambi climate	Turn on your A/C in the specified room, and run it in temperature mode	["Keep temperature at," "Which A/C"]
T1	Abode	Your abode system mode is changed	["Select mode"]
T2	Ambi climate	The air conditioner has been turned on or off	["Turn on/off," "Select A/C"]
T3	Ambi climate	Your device detects the temperature is below the threshold you specified	["Select A/C," "Temperature threshold"]

TABLE 2: Physical environment.

Physical environment	Source of trigger	Device or service of action	Keyword
Temperature	Temperature rises above	Air conditioner, heat pump	Temperature
Humidity	Humidity rises above	Air conditioner, heat pump, humidifier	Humidity
Brightness	Brightness higher than light dimmed to specific level	Dimmer, light	Brightness, light, dim
Air flow	Zone air flow rises above threshold	iZone	Air flow, airflow
Air quality	Air quality changes	Air purifier	Air quality
Sound	Sound detected says a specific phrase	Musaic, Nightingale, Woopla, GO, Android device, D-Link Siren	Sound, say, ask, play
Motion	Motion detected	Samsung robot vacuum	Motion, robot
Smoke	Smoke detected	\	Smoke

FIGURE 2: Overview of TapChain.

FIGURE 3: Example of dependency parsing.

TABLE 3: Explanation of the analysis results.

Sentence constituent	Analysis of Stanford CoreNLP
Predicate	Root, compound:prt, conj
Direct object	obj, obl, nsubj:pass
Indirect object	obl
Modifier	amod, compound

We do not extract the subject. Because we have deleted the beginning of the sentence in the text preprocessing, most of the sentences have become imperative sentences, and there is no subject. The valid subject only exists in the description of some triggers but, it does not exist in the description of actions.

4.3. Correcting. The Stanford CoreNLP sometimes makes analysis an error when analyzing sentences. We call this error predicate dislocation. For example, for the analysis of the sentence “add mixes to your favorite mixes,” the tool will mark “add” as the subject and “mixes” as the predicate, as shown in Figure 4. However, “add” should be the predicate and “mixes” should be the direct object.

This error has caused great obstacles to rule chain recognition, so we designed an algorithm to correct it. According to the observation, NLP tools only mark predicate and the direct object incorrectly but mark the indirect object and other sentence components starting from the indirect object correctly. Therefore, we modify the predicate and direct object and their relationship with other sentence elements. The correction algorithm is shown in Algorithm 1.

The algorithm first determines whether the syntax tree is empty and whether there is a subject. Then, the position of the subject and the part of speech of the subject are obtained in turn. If the part of speech of the subject is the original form of the verb and the position of the subject is at the beginning of the sentence, we can judge that it is a predicate dislocation. If the part of speech of our root word is the plural of verbs, it is further determined that the plural of nouns is wrongly judged as the third person singular. We update the part of speech of the root node to NNS (noun plural). Then, we swap the root node and the subject node and mark their relationship as a direct object. The corrected marking results are shown in Figure 5. Because there is only such predicate dislocation in the current dataset, our algorithm does not discuss other cases. In the future, we will update the algorithm if new marking errors appear.

4.4. Feature Extraction. Our goal is to determine whether an action can start a trigger. Wang et al. [4] only extracted features from verbs and objects, and their effect can be further improved. Therefore, we extract more features to help us identify the rule chain.

First of all, we use the lemma function of Stanford CoreNLP to transform the predicate and object words into word prototypes for feature extraction. Then, we extract features from five different aspects.

4.4.1. Predicate. Usually, the predicate is the core of a sentence; it has the function of connecting the preceding and the following. Through the predicate, we can know how to complete a task.

Same Predicate. The role of the predicate in a sentence determines its importance. Therefore, in a $\langle \text{action}, \text{trigger} \rangle$, we compare the predicate in action with the predicate in the trigger and construct a binary feature according to whether it is the same or not. In the description of trigger and action, there are parallel predicates. In this case, when we compare predicates, as long as one predicate is the same, we think that the predicate of action is the same as that of the trigger. We use the $f(s_p)$ to represent this feature, a_ps to represent all predicates in an action, and t_ps to represent all predicates in a trigger. We use a_p to

FIGURE 4: Wrong mark.

represent predicate from a_ps and use t_p to represent predicate from t_ps . We abstract the feature extraction as a formal expression, as follows:

$$f(s_p) = \begin{cases} 1, & \text{if } \exists a_p \in a_ps = \exists t_p \in t_ps, \\ 0, & \text{else.} \end{cases} \quad (1)$$

4.4.2. Object. Generally speaking, the object is the receiver of the predicate. Different objects, even if they have the same predicate, have opposite meanings. Therefore, it is equally important for us to identify the rule chain.

Same Direct Object. The direct object is the receiver of the predicate. We take it as a binary feature whether the direct object in action is the same as that in the trigger.

Same Indirect Object. The indirect object is a part affected. We also take it as a binary feature.

4.4.3. Modifier. Modifiers are used to modify nouns, which can describe some special properties of the modified nouns. Since we have extracted two types of objects, we also divide attributives into two types.

Same Modifier of Direct Object. Modifier of the direct object is used to modify the direct object. We take it as a binary feature.

Same Modifier of Indirect Object. Modifier of the indirect object is used to modify the indirect object. We also take it as a binary feature.

4.4.4. FieldList. On the IFTTT website, action and trigger have their own “FieldList,” which are “actionFieldList” and “triggerFieldList,” respectively. In “FieldList,” there is a lot of specific information. Therefore, we also extract three features from “actionFieldList” and “triggerFieldList.”

Exist Same Fields. We first traverse the “FieldList” of action and trigger respectively to find out whether there are the same fields. We take the number of same fields as a continuous feature.

Same Values in Same Fields. We also need to know if the values are the same in all the same fields. We take it as a binary feature.

The Number of Same Values. This is a supplement to the second feature. We calculate the number of the same personalized values. We take it as a continuous feature.

4.4.5. Environment. We first extract physical environmental factors from triggers. This is because our goal is to identify the rule chain. Only when the trigger can receive the physical environmental factors can the rule chain be formed. When we label manually, we do this work at the same time. We


```

Input:  $R$  (root of syntax tree)
Output:  $R'$  (root of corrected syntax tree)
(1) function FixPredicate( $R$ )
(2)   if ( $R == \text{NULL}$ ) and ( $\text{getSubject}(R) == \text{NULL}$ ) then
(3)     return  $R$ 
(4)   end if
(5)   subj_pos  $\leftarrow$  getSubjectPosition( $R$ )
(6)   subj_PoS  $\leftarrow$  getPartOfSpeech(subj_pos)
(7)   if (subj_PoS == VB) and (subj_pos == 1) then
(8)     root_PoS  $\leftarrow$  getPartOfSpeech( $R$ )
(9)     if root_PoS == VBZ then
(10)      subjNode  $\leftarrow$  getSubjectNode( $R$ )
(11)      root_pos  $\leftarrow$  getRootPosition( $R$ )
(12)      root_PoS  $\leftarrow$  NNS
(13)      SwapNode( $R$ , subjNode)
(14)       $R' \leftarrow$  subjNode
(15)       $R' \leftarrow$  setRelation( $R'$ ,  $R$ , obj)
(16)    end if
(17)  end if
(18)  return  $R'$ 
(19) end function

```

ALGORITHM 1: Correcting predicate dislocation.

FIGURE 5: Corrected mark.

extract eight kinds of physical environment factors, and at the same time, we extract keywords to extract features, as shown in Table 2.

For the physical environment, we can distinguish whether the indirect chain can be formed according to four characteristics.

Same Environmental Factor. First of all, the physical environment affected by an action needs to be the same as that detected by a trigger before the action can start the trigger. Therefore, we regard whether the physical environment affected by action is the same as the physical environment detected by trigger as the first physical environment feature, which is a binary feature. For the first three environmental factors, we extract them directly from the “actionFieldList” and “triggerFieldList.” But we cannot extract the hidden environmental factors of device effect and other environmental factors in the same way. Therefore, for a trigger, we extract it from “triggerDesc” by string matching and, for action, we summarize the environment affected by the device or service and the keywords to extract features in Table 2. According to the terms of the device or service and the keywords we

extracted, we judge whether the action will cause environmental changes.

The Types of Environmental Factors. Second, if the environmental factors are the same, we extract the types of environmental factors. We take it as a continuous feature. The values of this feature are from 1 to 8, representing 8 environmental factors in Table 2. 0 is used to indicate that the environment variables are different.

Same Value Range. Third, we need to judge whether the values of the physical environment are the same. When “higher, above, exceed, greater” appear in “triggerDesc,” we map the environment value range of trigger ($T(E_v_r)$) to $[T(E_v), \infty)$. When “lower, below, less” appear in “triggerDesc,” we map the environment value range of trigger ($T(E_v_r)$) to $(-\infty, T(E_v)]$. Then, we can get values from the “actionFieldList” ($A(E_v)$) and “triggerFieldList” ($T(E_v)$) and compare them. This is important in the first three environmental factors: temperature, humidity, and brightness because they all have certain values in trigger and action. We also define this feature as a binary feature. We use $f(s_v_r)$ to represent this feature. For the convenience of understanding, we abstract it as a formal expression, as follows:

$$\begin{aligned}
T(E_{-v-r}) &= \begin{cases} [T(E_{-v}), \infty) & \text{higher, above,} \\ (-\infty, T(E_{-v})], & \text{lower, below,} \end{cases} \\
f(s_{-v-r}) &= \begin{cases} 0, & A(E_{-v}) \notin T(E_{-v-r}), \\ 1, & A(E_{-v}) \in T(E_{-v-r}). \end{cases}
\end{aligned} \quad (2)$$

Same Trend. Finally, we also make it a feature whether the changing trend of the physical environment is the same. For other environmental factors, there may be an explicit numerical expression in the trigger, but we cannot find the description of numerical value in action. Besides, some devices will affect other physical environments. For example, we can get the temperature value of the air conditioner setting through the “actionFieldList,” but we cannot get the value that the air conditioner affects the humidity; we can only judge that it will lower the humidity. Therefore, we take it as a supplement to the second physical environmental feature. We define this feature as a binary feature. For the first five environmental factors, we define the trend as up or down according to the description. For the latter three environmental factors, we only define whether they exist.

4.4.6. Summary. We extract 9 binary features and 3 continuous features from five different perspectives. Among them, the five features of grammatical elements are based on NLP technology. The other features are more extracted by string matching and numerical comparison.

5. Evaluation

In this section, we evaluate the performance of TapChain. In the following evaluation, we use i7-5700HQ@2.70GHz quad-core processors, 8 GB memory.

5.1. Setup. We obtain 279,828 rules, 431 services, 1,470 triggers, and 896 actions. In the above data, only 150 services have both triggers and actions, so we only construct 5,528 $\langle \text{action, trigger} \rangle$ pairs with the same service and label them. It takes us about 60 hours to complete the task of labeling data. One of our collaborators reviews the labeled dataset and revises it. In the process of labeling data, we find that the positive and negative data are unbalanced, and the positive data only accounted for about one-tenth of all datasets. We do not have a large number of data, so we use SMOTE algorithm in imblearn [19] to balance the data. We randomly select the data of 80 services as the training set of the classifier, and the data of 70 services as the test set of the classifier.

In the choice of classifier, we do not consider too much. Since our task is a supervised binary classification task, we use four main supervised learning classifiers from scikit-learn [20] to train the model. The four classifiers are random forest, support vector machine (SVM) (RBF kernel), multiple perceptron, and logistic regression.

5.2. Performance. We use test sets to evaluate TapChain with accuracy, precision, recall, and F1-score [21]. In our opinion,

precision rate and recall rate are equally important in the recognition rule chain. Because, if the precision of the model is low, the future security policy based on the model will confuse users and seriously affect the user experience. If the recall rate is low, the identified rule chain will be incomplete, which will affect the security of users. Therefore, besides accuracy rate, precision rate, and recall rate, we use F1-score to evaluate TapChain. The results are shown in Table 4.

All evaluation indicators are calculated based on the following concepts. True positive (TP) is the number of rule chains correctly predicted. False positive (FP) is the number of rule chains incorrectly predicted. True negative (TN) is the number of correct recognition that will not form a regular chain. False negative (FN) is the number of incorrect recognition that will not form a regular chain. All is the number of all data. The accuracy rate (A), precision rate (P), recall rate (R), and F1-score ($F1$) are defined as follows:

$$\begin{cases} A = \frac{TP + TN}{All}, \\ P = \frac{TP}{TP + FP}, \\ R = \frac{TP}{TP + FN}, \\ F1 = \frac{2 \times P \times R}{P + R}. \end{cases} \quad (3)$$

According to the results, our model of identifying rule chain is better than that of Wang et al. [4]. TapChain performs better when using the same machine learning model. The reason why TapChain works better is that we extract more features and correct the results of NLP tool analysis. We not only extract additional semantic features from indirect objects but also extract additional environment and personalized features from “FieldList.” For the analysis error of the NLP tool, we design an algorithm to correct it, which can get the correct part-of-speech tagging and dependency parsing. Then, we can extract the correct features and improve the performance of TapChain. In addition, we split the sentences with conjunctions that are easy to cause trouble. When two sentences are compared and there is a same predicate, we think that the predicates of the two sentences are the same. It also improves the performance of TapChain.

Besides, they do not use the personalized data in “FieldList” when they label datasets, and in practical application, TapChain has more advantages in identifying rule chains. Because when users use these actions and triggers, they will set specific values. For example, for action “turn on light,” the user needs to set “which light” in “actionFieldList.” At this time, if there is a trigger “light turned on,” and the “which light” in the “triggerFieldList” is different from the “which light” set in the “actionFieldList,” then the action will not start the trigger.

To prove the rationality of our conjecture, we calculate the importance of every feature. Because the model based on

TABLE 4: Results (percentage).

	Model	Accuracy	Precision	Recall	F1-score
TapChain	SVM (RBF kernel)	89.7	87.8	92.1	89.9
	Random forest	89.5	87.9	91.7	89.7
	Multiple perceptron	88.1	87.8	88.5	88.1
	Logistic regression	89.9	88.2	92.1	90.1
Wang et al. [4]	SVM (RBF kernel)	80.2	—	90.7	—
	Random forest	85.7	—	88.2	—
	Multiple perceptron	86.8	—	88.6	—
	Logistic regression	83.1	—	84.4	—

logistic regression performs better, we calculate the importance of features according to logistic regression. In logistic regression, the importance of features is strongly correlated with the coefficients of features. Therefore, we reflect the importance of features through the coefficients of features. We can understand this more easily through the formal expression of logistic regression. The formal expression of logistic regression is as follows:

$$y = \frac{1}{1 + e^{(-\omega^T x)}}, \quad (4)$$

where x is a vector composed of different features, and ω is a vector composed of coefficients of different features. The solution process of ω is as follows:

$$\omega_{t+1} = \omega_t + \eta \left(y_n - \frac{1}{1 + e^{(-\omega^T x_n)}} \right) x_n, \quad (5)$$

where η is a constant, ω_0 is a random vector, and x_n and y_n are from a random training sample. Because ω is updated continuously, and finally an appropriate weight is calculated for every feature, the coefficient of every feature can represent the importance of the feature.

In addition, since scikit-learn officially provides the calculation method of feature importance in the random forest model, we also calculate the importance of each feature in the random forest, as shown in Figure 6.

To facilitate drawing, we use the abbreviation of feature as the coordinate axis. The relationship between abbreviations and features is shown in Table 5. For the feature importance of random forest, the closer the value is to 1, the more important the feature is. For logistic regression, the larger the absolute value of the coefficient of a feature, the more important the feature is. According to Figure 6, we find that the most important features come from predicates and direct objects, followed by “FieldList”-related features and environment-related features. The features extracted from indirect objects play a very limited role in the classifier. Our newly extracted features play an important role in the accurate recognition of rule chains.

5.3. Case Study. Because we extract more features, TapChain can identify the rule chains more accurately and more fine-grained. We find that there are some new threatening rule chains that cannot be recognized by the existing work. We give a relevant case study.

- (i) R1: if a trigger, then play audio from URL (Musaic).
- (ii) R2: if say a simple phrase (Google Assistant), then open the garage door you specify (Gogogate).

R1 is a virtual rule, while the action of R1 is real action and R2 is a real rule. However, the action of R1 is real action, and if this action is used, there will be a huge threat from this rule chain. An attacker can make a high-frequency signal with “open the garage door” coding and then insert the high-frequency signal into the audio file. Finally, the attacker uploads the audio file to the URL. This URL can be a song list subscribed by the victim or a public link used by the victim. When the victim starts R1, the audio file uploaded by the attacker will be played. Because the current sound receiving equipment cannot filter the high-frequency signal [22], Google Assistant will receive the high-frequency signal, “open the garage door,” from the audio. This causes R2 to be activated unexpectedly. Then, the attacker can commit further criminal acts, as shown in Figure 7.

Generally speaking, if an attacker wants to take advantage of the domino effect of the rule chain, they need to use a malicious rule to start the rule chain [4]. However, the existence of this rule chain makes the attacker does not even need to construct a malicious rule, and the attacker can directly use the rule chain.

In this rule chain, none of the predicate and object of R1’s action and R2’s trigger are the same, which can only be identified according to the features of environmental factors that we extracted. Previous work cannot recognize this kind of rule chain. However, it is very dangerous for users to construct this kind of rule chain unintentionally when using TAP.

6. Discussion

Usability. Our goal is to better identify the rule chain and then more accurately identify threatening rule chains for end-users. When users use TAP to automate devices or APPs, TapChain can help users identify rule chains. Users can decide whether to change the TAP rules according to their own needs. Because the recognition of the rule chain by TapChain is based on the description of the text, TapChain can be effective as long as the Internet of things platform or TAP platform has a complete description. In future work, we plan to use a user survey to obtain real datasets to evaluate and modify TapChain. In future work, an important aspect will be the construction of a security model. In this way, TapChain can directly present the threatened rule chain to users.

FIGURE 6: Feature importance. (a) Random forest. (b) Logistic regression.

TABLE 5: The abbreviation of feature.

Abbreviation	Feature
f_1	Same predicate
f_2	Same modifier of direct object
f_3	Same direct object
f_4	Same modifier of indirect object
f_5	Same indirect object
f_6	Same field
f_7	Same value in same field
f_8	The number of same value
f_9	Same environmental factor
f_10	The type of environmental factor
f_11	Same value range
f_12	Same trend

FIGURE 7: New potential threat.

Limitation. There are also some limitations to our work. First of all, manually labeling a dataset can lead to potential errors. Although we avoid errors by labeling and reviewing, manually labeling datasets can still be the source of the errors. Secondly, TapChain is difficult to identify some rule chains with time constraints. For example, one rule is to

turn on the washing machine, and another rule sends a message to me when the washing machine is turned off. Generally speaking, the washing machine stops automatically after running for a while. When the washing machine stops, a message will be sent. However, since TapChain cannot obtain the running time, it cannot accurately identify such rule chains. Finally, because TapChain's recognition of the rule chain is based on the description of the text, the performance of TapChain depends on the correctness of the text description. If there is a trigger or action description that does not conform to the actual behavior, TapChain will make a judgment error.

7. Conclusion

In this paper, we propose TapChain, a rule chain recognition model based on multiple features. We extract 12 features from 5 different aspects. We designed a correction algorithm to obtain more accurate NLP analysis results and make us extract features more accurately. Our evaluation shows that the recognition accuracy of TapChain is up to 89.9%, the precision is up to 88.2%, and the recall is up to 92.1%. Compared with the existing work, the accuracy rate is increased by 3.1%, and the recall rate is increased by 1.4%. In addition, TapChain's recognition of the rule chain is more fine-grained and we find a new kind of rule chain with a potential threat, which will cause serious consequences once exploited by attackers.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that the funding in Acknowledgements section did not lead to any conflicts of interest regarding the publication of this manuscript. Also, there are no any other conflicts of interest in the manuscript.

Acknowledgments

This work was supported by the National Key R&D Program of China (Grant Nos. 2021YFB3100700 and 2020AAA0107703), the National Natural Science Foundation of China (Grant Nos. 61 872 181, 62 032 025, 62 076 125, 62 071 222, U20A201092, U20B2049, U20B2050, and 61 702 236), the Natural Science Foundation of Jiangsu Province (Grant Nos. BK20200418 and BE2020106), the Guangdong Basic and Applied Basic Research Foundation (Grant No. 2021A1515012650), and the Shenzhen Science and Technology Program (Grant No. JCYJ20210324134810028).

References

- [1] B. Ur, E. McManus, M. P. H. Yong, and M. L. Littman, "Practical trigger-action programming in the smart home," in *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 803–812, Toronto Ontario, Canada, April 2014.
- [2] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: automated iot safety and security analysis," in *Proceedings of the 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 147–158, Boston, MA, USA, July 2018.
- [3] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: dynamic enforcement of security and safety policy in commodity iot," in *Proceedings of the Network and Distributed System Security Symposium*, February 2019.
- [4] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action iot platforms," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 1439–1453, London, United Kingdom, November 2019.
- [5] 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*, pp. 215–225, Osaka, Japan, September 2015.
- [6] B. Ur, M. P. H. Yong, S. Brawner et al., "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*, pp. 3227–3231, San Jose California, USA, May 2016.
- [7] C. Nandi and M. D. Ernst, "Automatic trigger generation for rule-based smart homes," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, pp. 97–102, Vienna, Austria, October 2016.
- [8] S. Yarosh and P. Zave, "Locked or not? mental models of iot feature interaction," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pp. 2993–2997, Denver Colorado, USA, May 2017.
- [9] W. Brackenbury, A. Deora, J. Ritchey et al., "How users interpret bugs in trigger-action programming," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, pp. 1–12, Glasgow Scotland, UK, May 2019.
- [10] V. Zhao, L. Zhang, B. Wang, S. Lu, and B. Ur, "Visualizing differences to improve end-user understanding of trigger-action programs," in *Proceedings of the Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–10, Honolulu HI, USA, April 2020.
- [11] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proceedings of the 2016 IEEE symposium on security and privacy (SP)*, pp. 636–654, IEEE, San Jose, CA, USA, May 2016.
- [12] M. Surbatovich, J. Aljuraidean, L. Bauer, A. Das, and L. Jia, "Some recipes can do more than spoil your appetite: analyzing the security and privacy risks of ifttt recipes," in *Proceedings of the 26th International Conference on World Wide Web*, pp. 1501–1510, Perth, Australia, April 2017.
- [13] Y. J. Jia, Q. A. Chen, S. Wang et al., "Contextlot: towards providing contextual integrity to appified iot platforms," in *Proceedings of the 4th Annual Network and Distributed System Security Symposium*, vol. 2, NDSS, San Diego, California, USA, March 2017.
- [14] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what? controlling flows in iot apps," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 1102–1119, Toronto, Canada, October 2018.
- [15] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1074–1088, Toronto, Canada, October 2018.
- [16] L. Zhang, W. He, J. Martinez, N. Brackenbury, S. Lu, and B. Ur, "Autotap: synthesizing and repairing trigger-action programs using ltl properties," in *Proceedings of the 2019 IEEE/ACM 41st international conference on software engineering (ICSE)*, pp. 281–291, IEEE, Montreal, QC, Canada, May 2019.
- [17] X. Mi, F. Qian, Y. Zhang, and X. Wang, "An empirical characterization of ifttt: ecosystem, usage, and performance," in *Proceedings of the 2017 Internet Measurement Conference*, pp. 398–404, London United, Kingdom, November 2017.
- [18] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of the 52nd annual meeting of the association for computational linguistics: system demonstrations*, pp. 55–60, Baltimore, Maryland, January 2014.
- [19] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: a python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 559–563, 2017.
- [20] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit-learn: machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [21] Z. H. Zhou, *Machine Learning*. Springer, Berlin, Germany, 2021.
- [22] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "Dolphinattack: inaudible voice commands," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 103–117, Dallas Texas, USA, October 2017.