Ruledger: Ensuring Execution Integrity in Trigger-Action IoT Platforms

Jingwen Fan*§, Yi He†§, Bo Tang*, Qi Li†, and Ravi Sandhu‡
*Information Security Laboratory, Sichuan Changhong Electric Co., Ltd.
†Institute for Network Sciences and Cyberspace & Department of Computer Science, Tsinghua University; BNRist
‡Institute for Cyber Security and Department of Computer Science, University of Texas at San Antonio
Email: *{jingwen.fan, bo.tang}@changhong.com, †{yihe2020, qli01}@tsinghua.edu.cn, ‡ravi.sandhu@utsa.edu

***Abstract*—Smart home IoT systems utilize trigger-action platforms, e.g., IFTTT, to manage devices from various vendors. These platforms allow users to define rules for automatically triggering operations on devices. However, they may be abused by triggering malicious rule execution with forged IoT devices or events violating the execution integrity and the intentions of the users. To address this issue, we propose a ledger based IoT platform called Ruledger, which ensures the correct execution of rules by verifying the authenticity of the corresponding information. Ruledger utilizes smart contracts to enforce verifying the information associated with rule executions, e.g., the user and configuration information from users, device events, and triggers in the trigger-action platforms. In particular, we develop three algorithms to enable ledger-wallet based applications for Ruledger and guarantee that the records used for verification are stateful and correct. Thus, the execution integrity of rules is ensured even if devices and platforms in the smart home systems are compromised. We prototype Ruledger in a real IoT platform, i.e., IFTTT, and evaluate the performance with various settings. The experimental results demonstrate Ruledger incurs an average of 12.53% delay, which is acceptable for smart home systems.**

I. Introduction

Smart Home Systems[1] and Internet of Things (IoT) devices have proliferated recently. Users use various devices, e.g. the smart lock of Samsung SmartThings [9], the smart watch of Garmin, as well as bulbs and smoke detectors in their smart home systems. All these facilities can be managed via trigger-action platforms, such as IFTTT [3], Microsoft Flow [5], and Zapier [12], and work together to provide different functionalities. Meanwhile, users set up rules in these trigger-action platforms to automate rule executions in their devices. Thus, these platforms are able to manage various devices from different vendors to automatically perform physical or virtual tasks according to the rules.

Meanwhile, smart home systems suffer severe security threats, such as privacy leakage [24] and violation of rule execution integrity [21]. In particular, violating rule execution integrity becomes a major concern of smart home security. For example, an attacker can generate a fake location [33] to the trigger-action platform, e.g., IFTTT, so that the platform will trigger the rule to open the smart lock even though nobody is at home. Similarly, if OAuth tokens in the smart home system are leaked or part of the system is compromised, an attacker can easily access user locations and remotely control IoT devices by invoking APIs [13], [20].

These security risks in smart home systems have been extensively studied in the literature [20], [32], [33], [37]. Unfortunately, as shown in Table I, none of the existing approaches can effectively protect the integrity of rule executions in these systems. For example, ESO [32] enforces situational access control for trigger-action platforms by encapsulating IoT remote APIs, which cannot avoid fake triggers in trigger-action platforms. DTAP [20] only enables secure token authorization for trigger-action platforms but cannot prevent malicious rule executions triggered by IoT platforms themselves. Similarly, Peeves [33] can only prevent event spoofing attacks by verifying physical events in IoT devices. Thus, the current trigger-action based smart home systems still lack protection for the integrity of rule execution.

In this paper, we systematically study vulnerabilities that enable malicious rule executions in smart home systems. The root cause of the vulnerabilities is that there does not exist any mechanism to ensure the authenticity and integrity of the messages transferred among different components in smart home systems. To this end, we propose Ruledger, a ledger-based IoT platform that can be integrated with existing platforms, to ensure the integrity of rule executions by enabling verification of the information in the smart home systems. In particular, Ruledger utilizes three ledger wallets to guarantee stateful records according to real information, and leverages smart contracts to enable protections for rule configuration and verification of the trigger events and action requests according to the stateful ledger records. Ruledger records all the action requests and events in the ledger as stateful transaction logs so that it can verify the authenticity of all information associated with rule executions based on the logs and the execution states in the ledger. Therefore, Ruledger can effectively prevent faked event sources of rule triggers and malicious action requests generated from the trigger-action platforms. The security analysis proves that Ruledger ensures the rule execution integrity under various attacks. We prototype Ruledger with the real IoT devices and the mainstream trigger-action platform, i.e., IFTTT, and evaluate the performance of Ruledger. The experimental results demonstrate that the overhead incurred by Ruledger is acceptable for real deployment. The increased delay is about 12.53%, and the throughput overhead decreases by about 6.5%, comparing to the baseline systems.

Our contributions can be summarized as follows.

- We systematically study the vulnerabilities of violating the

§The first two authors contributed equally to this work.

[1]In this paper, we focus on studying the security of typical IoT systems, i.e., smart home systems. We use IoT systems and smart home systems interchangeably.

TABLE I: Comparison with existing works.

	Trigger-Action API Attacks	Platform API Attacks	Device API Attacks	Privacy Violations	Event Spoof
Peeves [33]	✗	✗	✗	✗	✓
DTAP [20]	✓	✗	✓	✗	✗
ESO [32]	✗	✗	✗	✓	✗
Ruledger	✓	✓	✓	✓	✓

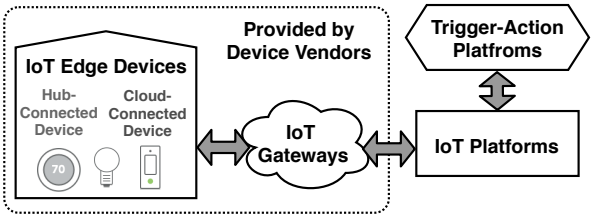


Fig. 1: The main components of smart home systems. The IoT gateways from device vendors serve as device brokers for IoT platforms. The trigger-action platforms may be third-party.

integrity of rule executions in trigger-action based smart home systems.

- We present Ruledger, a ledger based IoT platform, which verifies the authenticity of the information (events and the corresponding execution states) and ensures the integrity of rule executions.
- We develop state generation and verification algorithms built upon ledgers and wallets to guarantee that the states of the transactions in a smart home system are properly verified and submitted.
- We prototype Ruledger and the experiments with real-world IoT platforms demonstrate its performance.

II. Background

A. Smart Home Systems

Typically, smart home systems leverage trigger-action platforms to achieve inter-device automation by means of predefined rules. Such a trigger-action platform is operated either by a third party, e.g., IFTTT, or as part of a comprehensive IoT platform, e.g., Apple Homekit [1]. A user may configure the automation rules on the trigger-action platform specifying triggering conditions and actions of the corresponding devices. The data flow in a typical smart home system is illustrated in Fig. 1 which contains three components i.e., IoT edge devices, the IoT platform, and the trigger-action platform. The devices in a home may be from various vendors and connected to IoT platforms through the designated IoT gateways which are usually from different vendors. The IoT gateways interact with each other through trigger-action platforms as specified by user needs via OAuth APIs [7].

IoT Edge Devices and IoT Gateways. The IoT edge devices, including the corresponding IoT hubs providing wireless connection services, are the physical devices deployed in users' houses, and IoT gateways are deployed on the Clouds which provide remote APIs for device control. Some cloud-connected IoT devices with Internet access can directly connect to their online gateways, while some hub-connected devices that only have local wireless access (such as Zigbee, Z-Wave, and Bluetooth) need to connect to their IoT hubs. To remotely control these devices, IoT gateways provided by device vendors usually export APIs that are implemented via HTTP, WebSocket, or MQTT [6] for device access, and then the smart home systems can automate rule executions in the devices via these APIs.

IoT Platforms. IoT Platforms provide a programmable framework that wraps all the functionalities and APIs enabled in an IoT device. For example, SmartThings can set Device Handlers and SmartApps for IoT devices and leverage the Groovy scripts to define properties, e.g., event sending and event subscribing interfaces, for a specific device. Usually, IoT platforms, such as SmartThings and HomeKit, can manage both their own IoT devices and third-party devices. Particularly, the third-party devices first need to connect to their own gateways provided by the vendors and then communicate with the IoT platforms through the OAuth protocol [7].

Trigger-Action Platforms. Smart home systems allow users to set trigger-action rules to automate rule executions in devices. Some IoT platforms allow direct configuration of trigger-action rules (e.g. HomeKit), and many widely-used third-party trigger-action platforms, such as IFTTT [3], Zapier [12], and Microsoft Flow [5] can configure the rules as well. In this paper, for simplicity, we take the most popular trigger-action platform, i.e., IFTTT, as an example, which receives an HTTP request as a trigger event and then generates HTTP requests to trigger one or more actions.

Unlike IFTTT web apps [36], the IoT trigger-action rules are triggered and executed in IoT devices, and they are defined in forms of conditional expressions, i.e., if the trigger event happens then the actions will be executed in IoT devices. A trigger event can be a specific event or a status from the IoT device's operation, e.g. the door is opened or a user arrives home, and actions are a serial of operations on IoT devices. To set a trigger-action rule, users first need to choose an IoT operation to generate the trigger event and then set some operations as actions.

B. Distributed Ledgers

Distributed ledgers, such as blockchain systems, have been widely used to record various transactions across independent systems. A transaction is only ever stored when consensus has been reached by the involved parties. The light-weighted consensus algorithms, such as practical Byzantine Fault Tolerance (pBFT) [15] and raft [28], are extensively used to achieve data synchronization in distributed ledgers. In this paper, we leverage pBFT to achieve consensus among smart home systems, which is efficient in asynchronous environments.

Distributed ledgers utilize client-side agents (called wallets) to publish data in a decentralized database. The published data can also be user-defined executable codes called smart contracts [10] that can specify ledger functionalities with automatic execution on the records. Note that, a smart contract can be applied to various distributed ledgers that use different consensus algorithms.

III. Uncovering Vulnerabilities in Rule Execution

In this section, we systematically study the attacks that trigger malicious rule executions and violate the integrity of rule execution in smart home systems.

A. Threat Model

In this paper, we consider two types of typical attacks, i.e., API level attacks and platform/device compromise attacks, that trigger malicious rule execution on benign devices and incur the overprivilege problem in these devices. API level attacks access and manipulate the remote APIs of the systems, e.g., devices APIs, trigger-action API, or configuration APIs, by exploiting leaked OAuth tokens or vulnerable APIs, while platform or device compromise attacks directly compromise and control platforms or devices. Note that, we only consider a small part of devices and platforms being compromised, which can ensure correct rule execution in benign devices.

Particularly, we consider that trigger-action platforms are untrusted, which means attackers can construct API level attacks to arbitrarily create and trigger rules that will be triggered in benign devices. Moreover, attackers can leverage API level attacks [30] or compromise platforms to set malicious rules for benign devices in IoT platforms. Similarly, attackers can compromise edge devices or launch API attacks to trigger rule executions on benign devices. Note that, we focus on attacks that violate the integrity of rule execution, and thus do not consider other types of attacks such as denial of service (DoS).

B. Violating Integrity of Rule Execution

Unlike traditional client-server applications, IoT systems include diversified IoT services and heterogeneous devices. This makes the systems extremely complicated and vulnerable to the violation of rule execution integrity and abnormal rule executions, which therefore incurs the overprivilege problem in benign IoT devices. For simplicity, we use the example shown in Fig. 2 to illustrate how each component of IoT systems may be attacked. The corresponding example rule is *"when the user arrives home, enter home mode and turn on the lamps and open the yard's backdoor"*.

Now we identify various attacks in different components of smart home systems that can lead to violations of rule execution, i.e., malicious rule execution on benign devices.

Attacks against IoT Devices and IoT Gateways. IoT devices and IoT gateways may be compromised by attackers who can launch fake events attacks. As shown in Fig. 2, a rule is triggered by a fake event generated from a compromised device or a compromised IoT gateway that brokers messages

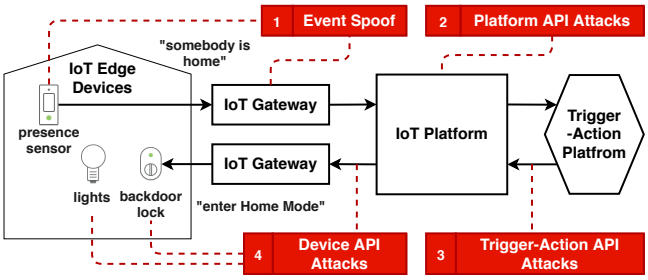


Fig. 2: Typical attacks against the smart home systems.

to and from the devices[2]. It is difficult to prevent attackers from compromising devices and arbitrary abuse of compromised devices. An attacker can easily conduct event spoofing attacks in devices (or gateways) to generate fake events, which trigger malicious rule executions in various benign IoT devices. Let us follow the example shown in Fig. 2. An event spoofing attack can easily spoof a state of the presence sensor which tricks the smart home IoT systems and the corresponding collaborative devices, such as lights, backdoor locks, etc., to activate the Home Mode when everyone is away.

In order to prevent the malicious devices from triggering automation rules for other devices, Peeves [33] utilizes an off-line event verification mechanism to ensure the authenticity of events, which uses the data of other sensors to check if their status have really changed after the event. However, this approach cannot be applied in trigger-action IoT systems that require instant rule execution after an event occurs.

Attacks against Trigger-Action Platforms. Third-party trigger-action platforms allow users to use trigger-action rules to automate IoT devices and usually, a token can be used by several APIs and involves multiple rules and devices. The token is often overprivileged and can be used for controlling all devices from a user. When the OAuth token is misused due to the platform or token being compromised, the attackers can abuse the rules to perform risky operations [14]. As is shown in Fig. 2, the untrusted trigger-action platforms may directly exploit the home mode actions without the trigger event of the user arriving home.

DTAP [20] aims to throttle the attacks by enabling a fine-grained short-lived token for each operation specified in a rule to ensure secure rule usage. In this setting, IoT platforms can issue new tokens to block malicious trigger requests.

Attacks against IoT Platforms. As IoT platforms are used to configure and execute rules, they need to interact with IoT devices and the trigger-action platforms. The IoT platforms should ensure the security of configuration APIs, which are used to set rules and manage user accounts and gateway APIs, as well as detecting event spoof attacks generated from the devices and preventing malicious requests from the trigger-action platforms. If attackers exploit the configuration APIs or the devices APIs, they can either trigger the malicious

[2]For simplicity, in this paper, we do not distinguish attacks against IoT devices and IoT gateways since the results have the same effect, i.e., spoofing events.

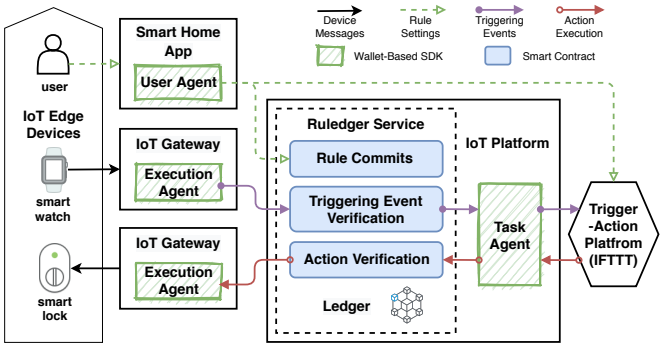


Fig. 3: The architecture of Ruledger

behaviors by modifying the trigger condition or trigger rule executions in the devices directly. Typically, rule execution is performed in the IoT platform, which invokes the Gateway APIs to control devices via the OAuth protocol, and all tokens of various devices are stored in the same place in the IoT platform. When the IoT platform is under the API level attacks (i.e., by using leaked tokens or vulnerable APIs) or is compromised, the attacker can easily abuse the device APIs.

In order to prevent API abuse, DTAP [20] develops an OAuth token protection mechanism for the trigger-action platform. However, it does not protect the IoT platform itself and cannot prevent rule tampering attacks that leverage vulnerable configuration APIs, e.g., manipulating user account settings, rule configurations, or the scripts that implement rules. Thus, DTAP is unable to prevent platform API exploitation that triggers malicious rule execution.

IV. Overview of Ruledger

In this section, we present an overview of Ruledger, which ensures execution integrity in smart home systems by establishing ledger based event and rule verification, and discuss the challenges in the design.

A. Overview

In this paper, we propose Ruledger, a distributed ledger-based IoT platform, which aims to ensure the integrity of rule execution by verifying the authenticity and consistency of the generated information in smart home systems. In particular, Ruledger utilizes wallet agents to enforce configurations and interactions among IoT Apps, IoT gateways, IoT platforms, and trigger-action platforms via transactions on the ledger. Different platforms can use APIs provided in the SDK of the wallet agent to realize interactions among different platforms. Ruledger can be integrated with existing third-party trigger-action platforms, which prevents various attacks of violating the rule execution integrity in smart home systems with trigger-action features. Note that, in Ruledger, we do not consider the API level attacks compromising the private keys in the wallets, which have been addressed by existing technologies [4], [8]. Fig. 3 shows the architecture of Ruledger. It consists of the rule commits module, the triggering event verification module, and the action verification module.

(i) The **rule commits** module guarantees all configurations are authentic. It consists of a rule commits smart contract in the IoT platform and wallet agents called user agents on the user side. This smart contract is used to set up device information and rule configurations in the ledger according to the information collected from the user agents that are integrated with users' IoT management apps. To configure a specific rule, users need to bind the device and set up the OAuth credential for connecting the corresponding gateway, and then set up trigger-action rules by choosing trigger operations and action operations in the app. All trigger-action rules will be committed to the ledger and exported to the third-party trigger-action platforms by the user agent.

(ii) The **triggering event verification** module verifies if the triggering event actually happens. It includes an event verification smart contract and wallet agents called execution agents in the IoT gateway. The smart contract verifies if triggering events are authentic by checking if the result of the trigger operation meets the trigger conditions according to the information reported from the execution agents. When a rule is triggered, each operation in the rule is associated with a standalone program, e.g., each operation in SmartThings devices is implemented as a device handler, in a groovy sandbox. The rule execution is performed by our execution agent if it meets the trigger condition. All triggering results are submitted to the ledger via transactions, and the execution logs that bind to a one-time key are recorded. The event records will be committed to the ledger if and only if the trigger transactions and the event logs are consistent, which prevents event spoofing attacks using fake execution logs.

(iii) The **action verification** module includes an action verification smart contract and task agents. The action verification smart contract verifies if the action should be executed by checking whether the action requests from the trigger-action platform are valid according to the ledger record and the transaction detail from the task agent. After the ledger receives an action request from the trigger-action platform, the action verification module checks if the condition of the rule is met based on the triggering event records on the ledger. Note that, when a new event record (discussed in ii) is committed to the ledger, the task agent reads the trigger record from the ledger and sends a trigger request to the trigger-action platform. The corresponding rule is triggered and the trigger-action platform sends action requests to trigger the action operations. These requests are converted to transactions by the task agent and the action verification module verifies if the previous corresponding trigger records are authentic. If the previous triggering event records are real and the trigger condition in the rule is met, the action operations will be executed. Meanwhile, these action operation transactions will be committed to the ledger and the execution agent updates with the records of these operations to verify subsequent rules.

B. Challenges

It is challenging to develop Ruledger since the interactions among IoT platforms, trigger-action platforms, and edge

devices may suffer various attacks exploiting rules, events or devices. We cannot simply rely on ledger transactions to achieve rule execution integrity.

- **Authenticity of Rule Configurations.** Trigger-action platforms and IoT platforms usually contain rules generated by various users and a user may also have multiple devices. It is difficult to prevent attackers or malicious users from generating malicious configurations. Ruledger should ensure rule executions on correct devices and prevent overprivileged operations, especially when the platform is compromised.
- **Preventing Fake Triggering Events.** The IoT platform should be able to verify if an event actually happens in the device and the rule execution condition is met after an event is triggered. However, it is difficult to achieve this because attackers can exploit devices with fake events by using the devices' cloud APIs. Moreover, the gateway's OAuth tokens should be protected and the event should be verified when a trigger operation occurs.
- **Preventing Malicious Transactions.** Ruledger should prevent malicious transactions generated by unauthorized rule administration in trigger-action platforms, which is difficult to achieve. Since the same OAuth tokens are always shared by multiple rules and devices, malicious transactions can be generated by sending action requests to the task agent with rewind attacks using reused or leaked tokens.

V. Design Details

In this section, we describe the design details of Ruledger and address the design challenges.

A. Rule Commits

We should prevent vertical privilege escalation [23] during committing rule configurations. Otherwise, attackers can abuse multiple users' rules and devices when the platform is compromised. To address this issue, we utilize a wallet based user agent, which ensures that users correctly configure devices and rules and manage their accounts. In particular, Ruledger adopts the role-based access control model for users to configure rules via the smart contract, which allows users with different permissions to correctly record all configurations in the ledger via the user agents. A sample smart contract for verification can be found in Appendix A-A. Similarly, all changes of configurations will be committed via ledger transactions, which are triggered by the user agent as well. To achieve this, different user agents with corresponding roles use their private keys indicating different permission levels, e.g., administrator's or normal user's private key. Finally, each transaction operation is verified by the smart contract, which checks the access control list based on the transaction signature signed by the corresponding private key of the user agent. Rather than traditional ways of setting rules in the trigger-action platform directly, users need to set up rules in our IoT platform and then the rules can be correctly committed to the ledger and exported to third-party trigger-action platforms such as IFTTT after the transaction is confirmed.

The ledger based IoT platform uses the pBFT [15] algorithm to perform the consensus process of recording updates, which ensures that the platform is secure unless more than one-third of nodes committing the records are compromised. Note that, when the user account leaks, attackers can only exploit the devices under the single account but cannot perform vertical privilege escalation. In particular, all rules and settings are committed to the ledger, which makes the rules and devices configuration tamper-proof.

B. Triggering Event verification

Ruledger prevents fake triggering events by verifying the execution logs of IoT devices and the event states recorded in the ledger. We utilize a ledger wallet called an execution agent to ensure that events recorded on the ledger are correct, which is used to protect device APIs and prevent privacy violation by verifying these records.

Various execution agents in the IoT gateway execute different operations for the devices and commit the corresponding information to the ledger, which can be used to verify that all operations are correctly executed and APIs are invoked with correct OAuth tokens. According to our key observation that operations specified in IoT rules are executed in separated procedures that run in different processes or machines. Our execution agent can be integrated into the corresponding IoT gateway of the devices, and then the OAuth tokens of the device HTTP APIs can be stored in the agent. Thus, the private data delivered by HTTP APIs will not leak into the IoT platforms and the trigger-action platforms, and these platforms only obtain the execution results, instead of the user information, e.g., user locations in the plain text. Similar to the situational access control [32], for a specific device, we develop prefab scripts (see Appendix A-B), that are similar to the code of SmartThings device handlers, which encapsulate the original HTTP APIs to implement operations for the device. Note that, if a rule includes multiple operations, we need to split the rule such that each script is associated with one operation. Before an operation is executed, an execution agent associated with the device needs to pull the corresponding scripts from the ledger to perform the executions.

All trigger operations specified in rules are registered after the rules are set up. They keep running in the execution agents that poll the trigger state by executing the operation and check the trigger conditions in a fixed interval (shown in Algorithm 1). When the trigger condition in a rule is met, the execution agent generates an event transaction that contains the rule information, the execution log's query key, and the corresponding checksum to the ledger. Upon receiving the information, the smart contract verifies the transaction and the corresponding execution log. If the verification succeeds, the event transaction is committed to the ledger and the task agent will generate a trigger request[3] to the trigger-action platform to trigger this rule (shown in Algorithm 2). LedgerVerify in

[3]Note that most IoT platforms, such as IFTTT, support push mode and does not poll the trigger state. After the rule is triggered, the IoT platform pushs the triggering event to IFTTT.

Algorithm 1: Check Trigger Condition

Result: Submit event transactions
// check the trigger conditions in execution agents
foreach $R_i \leftarrow Rules$ **do**
  trigger $\leftarrow R_i$
  (eid, log_key) $\leftarrow$ GenLogKey(R_i)
  $result \leftarrow$ ExecOperation(eid, trigger, log_key)
  if CheckTriggerCondition($result, R_i$) **then**
    // submit the event transaction
    log_sum $\leftarrow$ CheckSum($result$)
    event_log $\leftarrow$ (eid, log_key, log_sum)
    event_info $\leftarrow$ GenEventId(R_i)
    SubmitTransaction(event_info, event_log)

Algorithm 2: Triggering Event Verification

Input : An event transaction, T_i
Result: Gen trigger requests
// verify the event transaction and send trigger requests
(event_info, event_log) $\leftarrow T_i$
if LedgerVerify(event_info, event_log) **then**
  CommitTransaction(T_i)
  Cid $\leftarrow$ GenRandomness(event_info)
  // notify the task agent to send trigger requests
  TaskAgentSendRequest(event_info, Cid)

Algorithm 2 uses the event verification smart contract to verify the correctness of the transaction and stores it in the ledger if the transaction is authentic.

In order to defend against the event spoofing attacks on the devices, we utilize an execution log based event verification mechanism. Before a record transaction is accepted by the ledger, the ledger will verify the authenticity of the record by querying the execution log associated with the event. We leverage the existing logs generated by the device's cloud service and include a random key for each entry of the execution log, which ensures that only real devices can write the log. For a specific operation, the execution agent generates an execution id and a random key as the log query key pair (eid, log_key). As is shown in Fig. 4, this log query key pair is also sent to the device through the gateway (steps ❶, ❷). If the operation is successfully executed by the device, a new event_log that contains the log query key pair and the checksum of execution result is included in the log (step ❹). Also, the execution agent can obtain the same checksum (step ❸) with the execution results and submit the event_log as a transaction (step ❺ (see Algorithm 1). Finally, the triggering event verification smart contract in the ledger checks if the event_log associated with the transaction is the same as the record in the IoT log (step ❻). As all valid device operations are initiated by the execution agent with the unique log query key and the IoT gateway cannot directly access the log, Ruledger can easily identify malicious executions triggered by the gateway or devices by checking the verifiable event_log.

Therefore, the authenticity of triggering events can be ensured. Particularly, it is difficult for attackers to generate a fake event, which requires compromising both the device and the gateway, or both the log server and the gateway. Note that, we do not consider collusion attacks among several parts, which

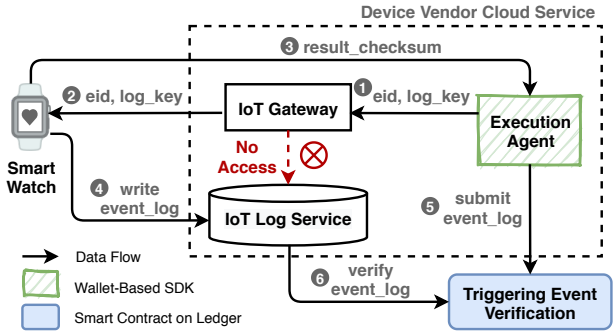


Fig. 4: The procedure of triggering event log verification

means the attacker can bypass the whole verification system. However, if APIs are manipulated or devices are compromised, the valid execution log associated with the operation execution will not be generated since the attacker does not have the key to generate the records, which is generated by the execution agents in the gateway.

C. Verifiable Action Execution in Ruledger

Now we describe how Ruledger realizes action requests verification, which aims to detect fake triggers and overprivileged transactions in the untrusted trigger-action platform. In the current trigger-action platforms, all rules and devices of a user share the same OAuth token, which makes a malicious trigger-action platform arbitrarily generates action requests. To address this issue, Ruledger utilizes fine-grained rule control on the trigger generation. A triggering event is sent to the trigger-action platform only when the rule trigger condition is satisfied. To distinguish different requests, we deliver the event identity eid and a random coin Cid that is generated by a pseudorandom function according to the event information. Note that, here we cannot use the existing token based solution [20] that leverages per-rule token for each triggering event, which cannot ensure the authenticity of triggers generated from the trigger-action platforms. To trigger a rule, the trigger-action platforms need to pass the correct eid and Cid to the task agent. Thus, attackers cannot forge the corresponding eid and Cid but only rewind the existing requests. However, after the actions are triggered, the execution records associated with the actions are also committed to the ledger by the execution agent. The rewinding requests will be rejected by the task agent and the ledger. Thus the malicious transactions are prevented and the attackers cannot exploit rules generated by the trigger-action platforms.

The pseudo-code of action verification is shown in Algorithm 3. After the ledger receives a request of an action transaction, the action verification smart contract (i.e., LedgerVerify, see Appendix A-A) decides if the transaction should be committed and executed. Unlike the existing smart home systems where the rule execution condition is only checked by the trigger-action platforms, in Ruledger, the execution of an action operation is confirmed according to the event record in the ledger. Thus, attackers cannot arbitrarily

Algorithm 3: Action Verification

```
Input  : A action transaction from the task agent, T_i
Result: Submit action transactions
(event_info, Cid) ← T_i
if VerifyRandom(event_info, Cid) then
    event_record ← LedgerQueryEvent(event_info)
    if LedgerVerify(event_record) then
        rule ← LedgerQueryRule(event_info)
        // submit action operations as transactions
        SubmitTransactions(rule.actions)
        // the execution agent execute the actions and record the
          execution log
        NotifyExecutionAgent(rule.actions)
```

trigger rules unless the trigger condition is met and the event transaction has been committed to the ledger. Moreover, with the tamper-proof event records in the ledger, we can ensure that such rules can only run once according to the matched triggering event in the ledger.

D. Discussion

Ruledger ensures the integrity of rule execution in the trigger-action based smart home systems that use conditions as triggers to execute operations specified in rules. Ruledger can validate all these conditions according to the state records in the ledger. Thus, it does not aim to ensure the execution integrity directly triggered by SmartApps that are implemented in Groovy scripts. Moreover, the wallet based agents in Ruledger use different private keys that should be protected by the existing key protection solutions [4], [11] or the trusted execution environments.

VI. Security Analysis

In this section, we analyze the security of Ruledger against the attacks.

API Level Attacks. Ruledger prevents API level attacks by ensuring the authenticity of users, rules, events, and triggers, and thus guarantees the execution integrity of smart home systems. All functionalities are performed via transactions that are executed and verified by smart contracts, and all execution records are stored in the ledgers. Thus, all fake events and fake actions triggered by APIs can be detected by Ruledger.

First, user accounts and rule configuration cannot be faked and manipulated. Ruledger utilizes a wallet private key to sign user accounts and rule configuration in the configuration synchronization phase, and records these information into the ledger via a consensus process. Any attacker cannot abuse API to inject fake users or rule configurations due to the lack of the key. Thus, all users and rule information cannot be faked and manipulated by API level attacks.

Second, Ruledger ensures that any events in smart home devices actually occur. Particularly, the execution of the operations can only be triggered by the trigger transactions on the ledger via the execution agents of Ruledger and each operation associated with event changes can only be triggered once through the execution agent. Thus, all execution logs associated with events are recorded in Ruledger. The attacks constructed by any malicious users, e.g., via a compromised user agent, cannot abuse any rule or trigger malicious execution of an operation specified in a rule because they cannot generate fake triggers that can be verified by Ruledger.

Third, triggers in Ruledger based smart home systems cannot be faked so that requests generated from the trigger-action platforms are authentic. In order to execute a rule in Ruledger, different parameters, e.g., event info and a temporary key Cid), are submitted to the trigger-action platform, and then the platform generates trigger requests that carry these parameters to trigger specific actions for IoT devices. As rule configurations and new rule execution tasks under execution are triggered by the execution agent of Ruledger in the devices, an attack against the trigger-action platforms can only exploit a single operation by generating and submitting fake parameters. Even if the attacker can inject the correct parameters to trigger the execution of an operation, the ledger in Ruledger can ensure that the operations of a task with correct parameters can only be executed once and the action operations are really executed based on the trigger records in the ledger. Note that, the trigger-action platform can only trigger an operation with valid parameters once when the trigger event is authentic and the operation is not overprivileged.

Platform Compromise Attacks. Ruledger can prevent the platform compromise attacks that may be constructed in each component. We will show how the execution integrity is ensured even under different platform compromise attacks.

If the trigger-action platform is compromised, attackers still cannot abuse the rules of various users to generate malicious requests because Ruledger will filter out such requests that are not recorded in the ledger of Ruledger. If the ledger based IoT platform is compromised, Ruledger can still ensure the rule execution integrity. Since rules and the corresponding operation records are triggered by smart contracts and generated via ledger transactions, the ledger nodes verify and execute the transactions by using the pBFT consensus algorithm [15] that can tolerate at most one-third of nodes being compromised. In the worst case, one-third of nodes are malicious, and they could prevent the committing of transactions or submit malicious transactions. The ledger in Ruledger can still reach consensus and commit the transactions successfully. Note that, if devices or the corresponding gateway are compromised, we cannot prevent the abuse of compromised devices. However, Ruledger can still prevent the attack of event spoofing since the attacker cannot inject fake logs of operation execution. The reason is that any fake events cannot produce fake logs in the log service of Ruledger as they are not invoked by the execution agent and cannot obtain the random log key to record the events.

VII. Performance Evaluation

A. Experiment Setup

We prototype Ruledger by using an open-source blockchain system [2]. The task agent, execution agent and user agent are implemented as web services with the Python Django framework invoking the wallet SDK to commit transactions and rules onto the ledger.

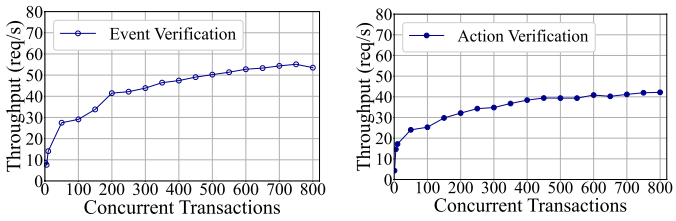
We evaluate the performance of each Ruledger module and the end-to-end system performance. In particular, we measure the latency and throughput of Ruledger modules and the integration of Ruledger with SmartThings and IFTTT. The prototype system is deployed in seven elastic cloud servers and each is configured with 4 core Intel Xeon CPU (3.10 GHz), 8G memory, and Ubuntu 20.04 OS. There are 4 blockchain nodes deployed on 4 servers. Note that in order to measure the throughput, we need to send multiple requests concurrently, which is constrained by the rate-limit mechanism in the SmartThings Device Handler and the IFTTT. Thus, we implement a skeleton device simulator and a trigger-action service similar to IFTTT Maker to measure the throughput of the system with and without Ruledger.

B. The Performance of Ruledger Modules

The main overhead of Ruledger is incurred by verifying rule triggers and rule execution, which are performed via smart contracts and the corresponding requests are converted to transactions. In this experiment, we measure the performance of the two modules and their impacts on the platforms. To accurately measure the performance, we use the task agent and execution agent to perform mock tests for transaction commit, and these agents do not interact with devices and trigger-action platforms but directly commit the mock transactions to the ledger. Note that, the delay of the task agent and the execution agent includes the delay incurred by other modules, such as request parsing and operation executions, which could have various delays corresponding to the detailed rules.

The Latency Incurred by Modules. To measure the latency of the transaction commit which is incurred by the trigger event verification smart contract module and the action verification smart contract module, we use an execution agent and a task agent to submit test transactions to the ledger directly. These transactions do not need to be triggered by requests from device gateways or trigger-action platforms so that we can get the real transaction commit time without the network latency. The average latency of the trigger event verification module is 32.45ms which includes all the procedures from a transaction being submitted by the execution agent to the verification of the smart contract and the transaction finally being committed to the ledger. And this time for the action verification module is 32.83ms. As the execution of a rule needs to commit the two transactions, the total latency of the smart contract modules is less than 70 ms which is acceptable as it is only 4.36% of the whole rule execution latency.

Throughput of Smart Contract Modules. Since a transaction is committed only when all the ledger nodes reach consensus via pBFT [15], the number of transactions that can be committed by the ledger in one second is limited. We submit transactions concurrently to test the ledger's transaction throughput threshold and check if it is the performance bottleneck of rule executions. We use the task agent and execution agent to submit different numbers of concurrent transactions and check how many transactions can be committed in a limited time.



(a) Triggering Event Verification (b) Action Verification

Fig. 5: The throughput of Ruledger verification modules

TABLE II: Comparison of average execution latency.

	SmartThings	Ruledger	Delay
End-to-end execution latency in average of 30 trials	1.403 s	1.604 s	12.53%

The throughput is plotted in Fig. 5. The concurrent transactions are submitted to the modules with valid test data for verification. By recording the start time and finish time of the transactions, we can measure the number of transactions processed per second. The throughput of the triggering event verification module is 43 TPS (transactions per second) and that of the action verification module is 55 TPS. Also, we evaluate how ledger nodes affect the throughput. The results show that the throughput is not significantly impacted by Ruledger. The results demonstrate the feasibility of Ruledger.

C. Performance of The Entire System

We measure Ruledger's overhead of rule execution in real-world deployments by integrating it with SmartThings and IFTTT and then testing the rule execution latency as well as the throughput of the whole system. We use the IFTTT Maker channel to trigger the *"heart rate alert"* rule and set simulated devices for a smart watch and a smart lock in the SmartThings WebIDE [9].

End-to-End Latency. To measure the latency of a round-trip from rule triggering to action execution, we deploy Ruledger as a middleware between the SmartThings platform and IFTTT and then execute rules under the protection of this middleware. And the baseline is implemented using the default usages of IFTTT that IFTTT directly controls the device handlers of SmartThings and do not use Ruledger as a middleware. We record the whole time of triggering and execution of the *"heart rate alert"* rule that is required by our ledger based version and the baseline.

The result is shown in Table II. The execution latency of the baseline includes time for two requests, the event request from a smart watch simulator of SmartThings to the IFTTT Maker channel, and the action request from the IFTTT to a smart lock device. By using logs in the SmartThings WebIDE, we can record the event request's sending time as the start time and the action request's receiving time as the end time to get the round trip time. However, it is impossible for users to filter out the network latency and the IFTTT's processing time since we can not get any log with an accurate time of the processing from the IFTTT. In the baseline system, the execution of the whole

TABLE III: The throughput of Ruledger and SmartThings with 2000 concurrent requests

	Baseline	Ruledger	Decrease
Throughput (req/s)	43.37	40.57	6.5%

rule finished in 1.403 s and in the system based on Ruledger, this time is 1.604s which includes time for interactions with the ledger and the time of sending requests. The overall latency of rule execution shows a 12.5% increase in the Ruledger version which is about 200 ms, comparing with the baseline.

The IFTTT's rule trigger [36] is not generated in real time as the default polling mode has a delay of seconds or minutes and the pushing mode incurs a delay of less than 1s. The latency of Ruledger does not affect the rule execution as the IoT rules do not need to be triggered immediately, which can tolerate a delay of several hundreds of milliseconds.

Throughput. As the SmartThings device simulators and the IFTTT have rate limit and cannot be triggered in a high frequency, we implement two simulated devices for smart watch and smart lock which can return the heart rate data or execute the unlock operation via HTTP requests, and we also implement a custom trigger-action service which works the same as IFTTT Maker to process unlimited requests. The rule could be triggered by sending requests with an abnormal heart rate, and we adjust the request number and frequency to test the throughput. For the Ruledger version, we use the execution agent and the task agent to convert requests to transactions and verify these transactions via ledger and smart contracts. In the baseline system, we just let the execution agent send origin requests to the task agent and do not interact with the ledger, as well as letting the task agent directly send the action requests to the execution agent to execute actions without the ledger's verification. In this way, we ensure that the network and machine status are the same in the two deploy versions.

To measure the throughput of rule processing, we construct 2,000 concurrent requests in the experiment. As shown in Table III, the system without ledger processes 43.37 requests of rule execution per second, and Ruledger handles 40.57 requests per second. The throughput is reduced by 6.5%, which is negligible.

VIII. Related work

Trigger-action Platform Security. DTAP [20] aims to provide action integrity for rule execution, which is most related to our work. It can prevent arbitrary rule executions from exploiting leaked OAuth tokens. In particular, it leverages coarse-grained XTokens in devices to protect rule-specific tokens, and verifies rule executions for action services. Unfortunately, it is unable to prevent attacks that use vulnerable APIs to generate fake events. Besides, DTAP can only constrain the ability of an attacker to steal a rule-specific token rather than preventing the attacker from exploiting rule execution. Ruledger well addresses this issue. Ruledger utilizes the tamper-proof feature of the ledger to verify the authenticity of the information and the integrity of rule execution. Moreover, The event spoofing attack[19] and active attacks[31], [22], [39], [38], [16] against trigger-action platforms is not well addressed in the literature. Our Ruledger well addresses these issues by automatically recording verifiable operation execution records in ledger with smart contracts.

Access Control for IoT System. IoT applications in IoT framework, e.g., SmartThings [9], use OAuth tokens to execute rules in target IoT cloud services, which suffers from the overprivileged issue due to the coarse-grained access control and lack of functionality isolation. Moreover, the exposure of original device APIs may incur privacy leakage. A number of studies focus on developing fine-grained access control mechanisms for IoT. For instance, FlowFence [18] utilizes a secure flow control mechanism to control sensitive data used by apps by executing device functions in sandboxes. FACT [34] uses access control list (ACL) registered by user to examine the privileges of applications. Situational access control [32] provides function wrapping for device APIs and uses unified IoT events, instead of revealing privacy data to IoT framework platforms or trigger-action platforms. These approaches can prevent overprivileged rule executions. Although these mechanisms heavily rely on OAuth tokens, device APIs can be easily abused if OAuth tokens are leaked. In order to prevent violation of execution integrity, Ruledger utilizes the device scripts in the ledger to verify the authenticity of the information, instead of using a token to execute rules.

Blockchain for IoT Security. Blockchain has been leveraged to protect IoT [26] [27] [17] [29] [25]. For example, a blockchain based smart home framework [26] records device management and device communication history as time ordered transactions and enforce these transactions according to embedded policies. Moreover, several blockchain based access control approaches [35], [27], [17], [29], [40], [25] have been proposed. Our Ruledger is orthogonal to these approaches. We can leverage these approaches to implement fine-grained access control in the ledger.

IX. Conclusion

In this paper, we propose Ruledger, a ledger based IoT platform, which is used to ensure the integrity of rule executions in trigger-action based smart home systems. Particularly, Ruledger utilizes applications built upon ledger wallets to honestly record information generated by smart home systems in the ledgers via ledger transactions, and smart contracts automatically verify the authenticity of the information associated with rule executions according to ledger transaction records. We prototype Ruledger with a real trigger-action platform and the experiment results with the prototype demonstrate that Ruledger incurs acceptable overhead for real deployment.

Acknowledgement

The work is supported in part by the National Key R&D Program of China under Grant 2018YFB1800304 and BNRist under Grant BNR2020RC01013. Bo Tang and Qi Li are the corresponding authors of this paper.

REFERENCES

[1] Apple Homekit. https://www.apple.com/ios/home/.
[2] FISCO-BCOS. https://github.com/FISCO-BCOS/FISCO-BCOS.
[3] IFTTT. https://ifttt.com/.
[4] Intel. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/key-protection-technology-paper.pdf.
[5] Microsoft Flow. https://flow.microsoft.com/en-us/.
[6] MQTT. https://mosquitto.org/.
[7] OAuth website. https://oauth.net/2/.
[8] Secure Wallet. https://www.ecomi.com/.
[9] SmartThings WebIDE. https://graph.api.smartthings.com/.
[10] The Ethereum Project. https://www.ethereum.org.
[11] Unbound Tech. https://www.unboundtech.com/solutions/blockchain-key-management/.
[12] Zapier. https://zapier.com/.
[13] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. Sok: Security evaluation of home-based iot deployments. In IEEE S&P, pages 1362–1380, 2019.
[14] L. Xing D. Zhao XF Wang D. Zou H. Jin B. Yuan, Y. Jia and Y. Zhang. Shattered chain of trust: Understanding security risks in cross-cloud iot access delegation. In USENIX Security, 2020.
[15] M. Castro and B. Liskov. Practical byzantine fault tolerance. In OSDI, page 173–186, 1999.
[16] H. Chi, Q. Zeng, X. Du, and J. Yu. Cross-app interference threats in smart homes: Categorization, detection and handling. In DSN, pages 411–423, 2020.
[17] D. Derler, K. Samelin, D. Slamanig, and C. Striecks. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. IACR Cryptology ePrint Archive, 2019.
[18] A. Rahmati D. Simionato M. Conti E. Fernandes, J. Paupore and A. Prakash. Flowfence: Practical data protection for emerging iot application frameworks. In USENIX Security, pages 531–548, 2016.
[19] J. Jung E. Fernandes and A. Prakash. Security analysis of emerging smart home applications. In IEEE S&P, pages 636–654. IEEE, 2016.
[20] J. Jung E. Fernandes, A. Rahmati and A. Prakash. Decentralized action integrity for trigger-action iot platforms. In NDSS, 2018.
[21] M. Balliu I. Bastys and A. Sabelfeld. If this then what? controlling flows in iot apps. In CCS, page 1102–1119, 2018.
[22] J. Martinez N. Brackenbury S. Lu L. Zhang, W. He and B. Ur. Autotap: synthesizing and repairing trigger-action programs using ltl properties. In ICSE, pages 281–291, 2019.
[23] P. Naldurg M. Monshizadeh and V. N. Venkatakrishnan. Mace: Detecting privilege escalation vulnerabilities in web applications. In CCS, page 690–701, 2014.
[24] L. Bauer A. Das M. Surbatovich, J. Aljuraidan and L. Jia. Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes. In WWW, page 1501–1510, 2017.
[25] D. Maesa and L. Mori, P.and Ricci. A blockchain based approach for the definition of auditable access control systems. Computers & Security, 84:93–119, 2019.
[26] D. Minoli and B. Occhiogrosso. Blockchain mechanisms for iot security. Internet of Things, 1:1–13, 2018.
[27] Os. Novo. Blockchain meets iot: An architecture for scalable access management in iot. IEEE Internet of Things, 5(2):1184–1195, 2018.
[28] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In {USENIX} ATC, pages 305–319, 2014.
[29] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman. Fairaccess: a new blockchain-based access control framework for the internet of things. SCN, 9(18):5943–5964, 2016.
[30] A. Bates Q. Wang, WU. H and C. Gunter. Fear and logging in the internet of things. In NDSS.
[31] W. Yang S. Liu A. Bates Q. Wang, P. Datta and Carl A. Gunter. Charting the attack surface of trigger-action iot platforms. In CCS, pages 1439–1453, 2019.
[32] V. Shmatikov R. Schuster and E. Tromer. Situational access control in the internet of things. In CCS, page 1056–1073, 2018.
[33] S. Eberz S. Birnbach and I. Martinovic. Peeves: Physical event verification in smart homes. In CCS, page 1455–1467, 2019.
[34] J. Kim B. Cho S. Lee H. Kim S. Lee, J. Choi and J. Kim. Fact: Functionality-centric access control system for iot programming frameworks. In SACMAT, pages 43–54, 2017.
[35] B. Tang, H. Kang, J. Fan, Q. Li, and R. Sandhu. Iot passport: A blockchain-based trust framework for collaborative internet-of-things. In SACMAT, pages 83–92, 2019.
[36] Y. Zhang X. Mi, F. Qian and XF. Wang. An empirical characterization of ifttt: Ecosystem, usage, and performance. In IMC, page 398–404, 2017.
[37] YH. Lin XF. Wang B. Ur XZ. Guo Y. Tian, N. Zhang and P. Tague. Smartauth: User-centered authorization for the internet of things. In USENIX Security, pages 361–378, 2017.
[38] G. Tan Z. Celik and P. McDaniel. Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In NDSS, 2019.
[39] P. McDaniel Z. Celik and G. Tan. Soteria: Automated iot safety and security analysis. In USENIX ATC, pages 147–158, 2018.
[40] G. Zyskind and O. Nathan. Decentralizing privacy: Using blockchain to protect personal data. In IEEE S&P Workshops, pages 180–184, 2015.

APPENDIX A
RULE AND SMART CONTRACT SAMPLES IN RULEDGER

A. A Sample Smart Contract for Information Verification

```
pragma solidity ^0.4.24;

contract Executor is LibUtils, ORM{
  function ledgerVerifyTrigger(int usr_rule_id, int
  usr_id, int rule_id, string rule_name, int step_id
    ) public returns(int) {
    // check rule execution access
    int memory usrItem = verifyUsrRule(usr_rule_id,
    usr_id, rule_id, rule_name);
    if (usrItem == RES_ERR) {
      return ERR_USER_VERIFY_FAILED;
    }
    // triggering event verification
    string[] memory conKeys = new string[](2);
    int[] memory conVals = new int[](2);
    conKeys[0] = "tRule_id";conKeys[1] = "tStep_id";
    conVals[0] = usrItem; conVals[1] = step_id - 1;
    string[] memory trEvts = new string[](3);
    trEvts[0] = "tTask_id"; trEvts[1] = "tStep_id";
    trEvts[2] = "tResult";
    int[] memory rets = selectEntry(trEvtTblName,
    conKeys, conVals, trEvts);
    // check if the trigger record exist
    if (rets.length <= 0 || rets[1] != step_id - 1
    || rets[2] != RES_OK) {
      return ERR_TRIGER_VERIFY_FAILED;
    }
    return RES_OK;
  }
}
```

B. Defined Rules in Ruledger

```
def alert_on_heart_rate(device_id, token):
  data = RPC_CALL(HEAR_RATE_API, device_id, token)
  if data.success:
      heart_rate = data.get("heart_rate")
      return heart_rate <= MIN_RATE or
          heart_rate >= MAX_RATE
  return False

def open_door_operation(device_id, token):
  res = RPC_CALL(SMARTLOCK_UNLOCK, device_id, token)
  return res.success

title = "alert on heart rate"
TRIGGER_OPERATIONS = [(alert_on_heart_rate,
    device_info1, OP_AND)]
CONDITION = IF_TRUE # lambda v: v is True
ACTION_OPERATIONS = [(open_door_operation,
    device_info2, OP_AND)]
RULE_DEFINE(title, TRIGGER_OPERATIONS, CONDITION,
    ACTION_OPERATIONS)
```