2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)

HomeScan: Scrutinizing Implementations of Smart Home Integrations

Kulani Mahadewa*, Kailong Wang*, Guangdong Bai†, Ling Shi*, Jin Song Dong*† and Zhenkai Liang*

National University of Singapore*, Griffith University†

Abstract—**A key feature of the booming smart home is the integration of a wide assortment of technologies, including various standards, proprietary communication protocols and heterogeneous platforms. Due to customization, unsatisfied assumptions and incompatibility in the integration, critical security vulnerabilities are likely to be introduced by the integration. Hence, this work addresses the security problems in smart home systems from an *integration* perspective, as a complement to numerous studies that focus on the analysis of individual techniques. We propose HomeScan, an approach that examines the security of the implementations of smart home systems. It extracts the abstract specification of application-layer protocols and internal behaviors of participants, so that it is able to conduct an end-to-end security analysis against various attack models. Applying HomeScan on three extensively-used smart home systems, we have found twelve non-trivial security vulnerabilities, which may lead to unauthorized remote control and credential leakage.**

I. Introduction

Enabled by the various intelligent Internet of Things (IoT) techniques, the smart home paradigm has been significantly changing the lifestyle of its users. New convenient facilities, such as smart lighting systems, smart TVs and security alarm systems, are becoming ubiquitous. Along with the booming growth of smart home, security incidents have been continually observed [1], [2]. Researchers have made efforts to address security issues in smart home systems [3], [4], with focus on several aspects, ranging from radio communications, networking, operating systems, middleware, and protocols, to backend cloud services.

In this work, we investigate security of smart home systems from an *integration* perspective. Our motivation is out of such a key observation—to realize a "smart" automated home, it is essential that multiple subsystems are integrated. The controls are typically initiated from the handheld devices such as smart phones, transmitted over wireless channels such as Bluetooth, ZigBee and Wi-Fi, forwarded by intermediate relays such as gateways, and web-based service portals, and finally executed by the end devices such as bulbs and locks. Due to the involvement of such a wide assortment of technologies and devices (usually from diverse manufacturers), it becomes challenging to coordinate them into a secure system. The challenge may be attributed to at least the following two factors.

- **Incompatibility.** Since diverse standards are enforced, there may be incompatibilities among the subsystems. For example, in the Philips Hue system that we have analyzed, the authentication between the bulb and the hub is through the Touchlink Commissioning (TLC) over ZigBee, while that between the hub and the control app is through a customized authentication over Wi-Fi. Once these three are integrated, due to the incompatibility between the two mechanisms, there is no way for the bulb to authenticate the control app. This allows a malicious app which has infected the mobile phone that the control app is installed on to acquire control over the bulb.
- **Invalidated Assumptions.** A developer or manufacturer may make assumptions (e.g., trust relation, message format and correct sequence of API calls) when using the interfaces provided by other parties. If the assumptions are invalid, the interfaces may be used in an insecure way. For example, in the same system above, the manufacturer of the hub actually assumes the LAN is secure, whereas this assumption may not be true if a malicious app has been installed on the user's mobile phone.

We present an approach named HomeScan, which scrutinizes security of the implementations of smart home systems. It extracts the application-layer protocols and security-relevant internal behaviors of each subsystem (or protocol participant) from the implementations. Through this, it can derive a unified abstraction of the end-to-end system to flatten the difference of the protocols employed by each participant. The challenges yet stem from the partial availability of the implementations. First, the source code is seldom visible, although the executable of the control app (from the app market), the firmware extracted from devices, and SDKs provided by vendors, are available for analysis. Second, the cryptographic protocols are used among the participants, so that the communication is blurred to us, even though we are able to capture the exchanged traffic. To alleviate these challenges, HomeScan uses a hybrid analysis including *dynamic testing*, *whitebox analysis* and *trace analysis*. The dynamic testing executes test cases, and captures communication traffic and execution traces; the whitebox analysis identifies semantics by analyzing the program that is available; the trace analysis infers the association relation between a value of unknown semantics and a participant, a session or a value whose semantics has been identified.

HomeScan uses labeled transition systems (LTSs) [5] that have been extensively used to model and reason various sys-

978-1-5386-9341-4/18/$31.00 ©2018 IEEE
DOI 10.1109/ICECCS2018.2018.00011

tems to represent the extracted specification. An LTS describes the execution of a particular participant, including its internal behaviors (e.g., generating a nonce and validating a digital signature) and communication behaviors (e.g., sending and receiving a message). At this abstract level, the security reasoning can ignore the heterogeneity of underlying protocols, but focus on the logic that is implemented by the system. Using this abstraction, reasoning security properties of the whole integration becomes effective, and we show that most of the properties specific to the smart home systems can be reduced to *reachability* checking.

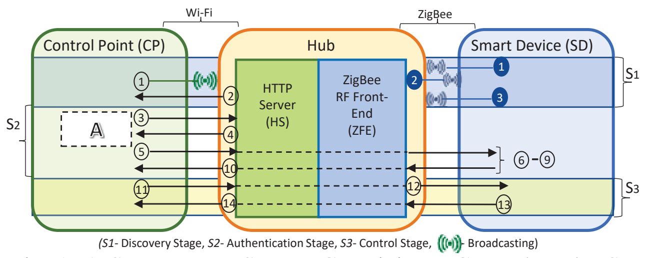
It is obvious that obtaining the complete or sound specification is almost infeasible. HomeScan focuses on extracting as precise specification as possible, whereby it can identify security issues. We prototype HomeScan and apply it to three extensively-used smart home systems, including Philips Hue, LIFX, and Chromecast. It manages to identify twelve security vulnerabilities.

Contributions. To summarize, we make the following main contributions in this paper.

- **Specification Extraction Techniques.** We propose hybrid techniques to extract specifications from the implementations of the smart home systems. Our evaluation of real-world systems demonstrates that the extracted specification is precise enough to identify significant security issues.
- **Vulnerability Identification Techniques.** We have modeled a set of practical attacks to facilitate the vulnerability identification techniques based on LTS representations. We reduce the vulnerability identification to traditional reachability analysis on LTS.
- **Practical Results.** We apply HomeScan to real-world systems and successfully identify twelve non-trivial security vulnerabilities from them. The supporting materials are published online for future research [6].

II. Background and Overview

In this section, we brief typical specification of smart home systems from the *integration perspective*, and provide an overview on the security properties and attack models in the vulnerability identification of smart home systems.



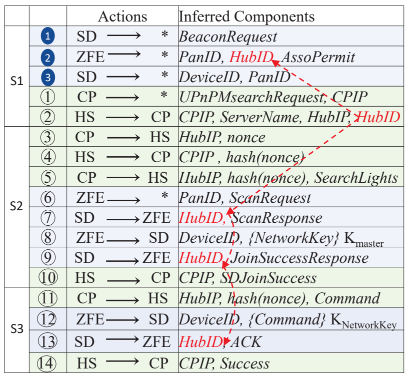
Wi-Fi
ZigBee
Control Point (CP)
Hub
Smart Device (SD)
HTTP Server (HS)
ZigBee RF Front-End (ZFE)
A
S1
S2
S3
⑥-⑨
(S1- Discovery Stage, S2- Authentication Stage, S3- Control Stage, Broadcasting)

Fig. 1: A Smart Home System Containing a CP, Hub and a SD

A. Overview of a Smart Home System Specification

In order to facilitate our LTS construction, we abstract a generic system architecture from several smart home systems popular on the market, such as Samsung SmartThings [7] and HomeGenie[8]. In our abstraction, a smart home system consists of three subsystems, i.e., a control point (denoted by *CP*) which interacts with the end users and issues the controls, several smart devices (denoted by *SD*) which are operable electronic end devices, and several relays (denoted by *hub*) which bridge the communications. Covering from configuration to control, the working procedure of the end-to-end smart home system is divided into three stages, i.e., *discovery*, *authentication* and *control*.

In Fig. 1, we show an example of this architecture, and its specification which HomeScan aims to extract is listed in Fig. 2. In this example, the CP is an Android app which supports HTTP protocol over Wi-Fi, and the SD only supports the ZigBee protocol. Therefore, the *hub* further includes an HTTP server (denoted by *HS*) and a ZigBee front end (denoted by *ZFE*) to facilitate the communication between the HTTP-based CP and ZigBee-based SD. In a nutshell, the system works as follows.



		Actions	Inferred Components
S1	❶	SD $\longrightarrow$ *	*BeaconRequest*
	❷	ZFE $\longrightarrow$ *	*PanID, HubID, AssoPermit*
	❸	SD $\longrightarrow$ *	*DeviceID, PanID*
	①	CP $\longrightarrow$ *	*UPnPMsearchRequest, CPIP*
	②	HS $\longrightarrow$ CP	*CPIP, ServerName, HubIP, HubID*
S2	③	CP $\longrightarrow$ HS	*HubIP, nonce*
	④	HS $\longrightarrow$ CP	*CPIP , hash(nonce)*
	⑤	CP $\longrightarrow$ HS	*HubIP, hash(nonce), SearchLights*
	⑥	ZFE $\longrightarrow$ *	*PanID, ScanRequest*
	⑦	SD $\longrightarrow$ ZFE	*HubID, ScanResponse*
	⑧	ZFE $\longrightarrow$ SD	*DeviceID, {NetworkKey}* K_{master}
	⑨	SD $\longrightarrow$ ZFE	*HubID, JoinSuccessResponse*
	⑩	HS $\longrightarrow$ CP	*CPIP, SDJoinSuccess*
S3	⑪	CP $\longrightarrow$ HS	*HubIP, hash(nonce), Command*
	⑫	ZFE $\longrightarrow$ SD	*DeviceID, {Command}* $K_{NetworkKey}$
	⑬	SD $\longrightarrow$ ZFE	*HubID, ACK*
	⑭	HS $\longrightarrow$ CP	*CPIP, Success*

Fig. 2: Inferred Specification for the Example in Fig. 1

- **Discovery Stage** (denoted by $S1$ in Fig. 1 and Fig. 2). The CP searches for the *hub* and pairs with the HS (step ① & ②). After searching existing ZigBee networks to join, the SD discovers the network created by ZFE (step ❶ -❸).
- **Authentication Stage** (denoted by $S2$). The CP and the SD authenticate themselves to the *hub*. To this end, the CP and the HS use a customized authentication (step ③-④), while the SD and the ZFE use the TLC of ZigBee Light Link (ZLL) [9] profile (step ⑥-⑨) after the CP requests the *hub* to find and connect SDs in the vicinity (step ⑤). Once the SD is authenticated by the ZFE, the HS sends a success response to the CP (step ⑩).
- **Control Stage** (denoted by $S3$). The CP controls the SD which is connected to the *hub* by sending control commands to the HS (step ⑪-⑭). Once receiving a command, the *hub* converts it to a ZigBee packet and sends it to the SD; upon the ACK received at the ZFE, the HS sends a success response to the CP.

B. Security Properties and Attack Models

Security Properties. Our approach analyzes the security properties including data security (i.e., data confidentiality and integrity), association security, and access security (i.e., authentication and authorization), given that various work has shown the importance of these security properties to IoT [10], [11], [12]. These properties are detailed in Table I.

Attack Models. The common threats to a smart home system are unauthorized access, manipulation by malicious partici-

TABLE I: Security Properties

Property	Security Property Description
Data Security	The property ensures that the data transmitted in a smart home system should be delivered to the intended participants without being revealed or altered by the unauthorized participants.
Associa -tion Security	The association between the user/device behaviors and the corresponding encrypted messages should not be revealed by exploiting meta data of the encrypted messages such as the packet size, the message source/destination addresses, etc.
Access Security	The property ensures that all participants in a smart home system can verify the identities of their communicating participants, and only the authorized participants are granted access to services and information.

pants [13], [14], and vulnerable settings of wireless communications [15]. Hence, we consider both malicious participants and network attackers, whose capabilities are described in Table II. Malicious participants are able to sabotage the access security properties by pretending to be honest ones to collect information (e.g., identities of other participants and session keys) illegally or send unauthorized commands to honest participants. Network attackers are able to eavesdrop, intercept and modify messages within the local network (e.g., Wi-Fi and ZigBee) in which the attacker resides or over the Internet.

TABLE II: Attack Models and Capabilities

Attack Model	Attack Capability Description
Malicious Participa -nts	**Malicious CPs** are able to manipulate victim *hubs* and SDs over the same local network or Internet by sending unauthorized commands.
	Malicious *hubs* are able to manipulate the victim SDs in the vicinity by sending unauthorized commands.
	Malicious SDs are able to capture the sensitive information (e.g., identity and address of the *hub* or CP) and possibly even take control of the victim hubs and other SDs in the vicinity.
Network Attacker	**Eavesdropping.** The attacker is able to obtain crucial information (e.g., session keys and the identity of the *hub*) by eavesdropping.
	Intercepting and Modifying Control Activities. The attacker is able to manipulate the system behavior by either replaying or modifying control commands such as ON/OFF of SDs, casting a video and changing light color.
	Intercepting and Modifying Administration Activities. The attacker is able to replay or modify the administrative commands such as device authentication/removal/reset, possibly causing functional disruption including Denial of Service.

III. HomeScan Overview and Prerequisites

In this section, we present an overview of HomeScan.

A. HomeScan *Overview*

HomeScan uses a set of techniques for specification extraction and vulnerability identification. It takes the following inputs.

- **Implementation of the Smart Home System.** A runnable setup of the smart home system and a set of programs (*PS*), including available source code, libraries, and binaries of participants are input to the HomeScan.
- **Test Cases.** A set of test cases (*TC*) is required to trigger the functionality of the smart home and at least one test case is required to initialize trace capturing component. A test case should include steps to discover, authenticate and control of the system. Each test case corresponds to a configuration of the system. Configurations refer to the participants (e.g., CP, SD and *hub*) of the system and the different users (e.g., admin, general user and guest).
- **Initial Knowledge.** Initial knowledge (*IK*) is represented as a 3-tuple (P, CH, KV), where P is the set of participants of the input system; CH is the set of channels used for communication among participants; KV is the set of knowledge required to execute the TC.

As shown in Fig. 3, HomeScan includes three major components including *trace capturing* and *pre-processing*, *specification extraction* and *flaw identification*.

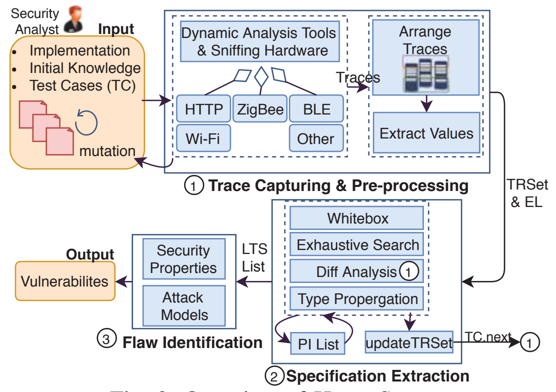


Fig. 3: Overview of HomeScan

Trace Capturing. The first step of the HomeScan is to capture the trace of the system under analysis by executing the initial test case. It captures two types of traces, i.e., traffic traces and execution logs. HomeScan uses existing sniffers to capture the traffic traces, and records the execution of the participants, whenever instrumentation can be done. In addition, HomeScan generates new traces by mutating the values (e.g., HTTP header values or HTTP parameters) from the captured traces, after executing each test case.

Pre-Processing. Pre-processing takes the set of captured traces as input and aims to generate a set of transactions (defined soon). A captured trace is a sequence of messages, containing the exchanged data between two or more participants. HomeScan first merges the traces in chronological order and then extracts the values from the traces. Based on the underlying protocols, HomeScan extracts data referring to their standard message formats. The extraction is done using keyword (e.g.,"host" in an HTTP request) searching, pattern matching and string splitting with delimiters (e.g.,"&").

Specification Extraction. The objective of this step is to generate LTS representation of the system, given the transactions generated from the pre-processing component. We propose a hybrid extraction technique including *whitebox analysis* and *trace analysis* for the specification extraction. The extracted specification is represented by LTS. In Section IV, we detail the specification extraction component.

Flaw Identification. In this step, we propose a verification algorithm to check IoT-specific security properties of the LTS representation against predefined attack models. Essentially, the verification algorithm is a reachability analysis. It can apply any of classic searching algorithms (e.g., DFS and BFS) on the generated LTS to search the reachability of a bad state wherein the security property is violated. In Section V, we detail our verification algorithm.

B. Prerequisites

In order to bridge the semantic gap between the low-level traces and the high-level LTS, we introduce several intermediate data structures to maintain the information required to generate an LTS.

Transactions. A transaction (*TR*) is a preliminary abstraction of one round of information exchange. We represent it as a 5-tuple $(id, se, R, EVSet, BR)$, where *id* is the transaction ID, $se \in P$ is the sender, $R \subset P$ is the set of receivers, and $EVSet = \{EV_1, EV_2, ..., EV_\alpha\}$ is the set of values (total number α) extracted from the message exchanged in the *TR*. Each EV_i is a 3-tuple (v, t, id) where *v* is the value, *t* is its type, and *id* is the value ID. The transaction also includes branch information (*BR*), which is defined soon.

To represent the output of the pre-processing component, we propose a transaction set denoted by $TRSet = \{TR_1, TR_2, ..., TR_\beta\}$ where β is the total number of transactions. Additionally, the ordered execution logs are output as a sequence *EL*.

Branch Information. Each transaction *TR* includes a branch set (denoted by *BR*), which is a set of transaction IDs that represent the transactions branching from the current transaction. There are three types of branches, i.e., options, self-recursions, and sequence-recursions. An option branch is either labeled as an option in the test case, resulted from test case mutation or resulted from configuration changes. HOMESCAN identifies self-recursions or sequence-recursions when data of a single transaction or data of a sequence of transactions are repeated in the trace respectively.

Types. For each extracted value $EV \in \bigcup EVSet_i$ $(1 \le i \le \beta)$, HOMESCAN attempts to identify a type (*t*) during the specification extraction. HOMESCAN defines two categories of types, i.e., *primitive* and *domain-specific*. The primitive type can be an integer, boolean, or string. The domain-specific type can be any of network address (used in ZigBee-like protocols), IP address, MAC address, username, password, encryption key, etc. During pre-processing, HOMESCAN assigns a primitive type to $EV.t$ and updates it to a domain-specific type (which is more precise) when more information is inferred.

The domain-specific types are formalized as *terms* (denoted by *T*). *Terms* are categorized into three subsets, i.e., *Constants* (denoted by *C*), *Functions* (denoted by *F*), and *Variables* (denoted by *V*), such that $T = C \cup F \cup V$. Ground terms are terms that only contain constants and functions. Variables are terms that are not ground. Table III lists part of the *Functions terms* use by HOMESCAN, and the full list is included in the technical report [6].

TABLE III: Function Terms

Function Term (F)	Definitions	Meaning
$\textbf{senc}(msg, k)$	$msg \in T$; symmetric key $k \in T$	ciphertext
$\textbf{hash}(msg)$	$msg \in T$	hash value

Actions. A label of an LTS is an action which can be either a communication or a local action. The actions which send and receive messages with other participants are communication actions, and the actions that execute local behaviors of each participant are local actions. Table IV lists several actions used by HOMESCAN.

TABLE IV: Communication and Local Actions

Type	Action	Definitions	Meaning
Comm.	$send(ch, msg)$	$ch \in C$; $msg \in T$	sending a message via channel *ch*
	$receive(ch, x)$	$ch \in C$; $x \in V$	receiving a message and storing in *x*
Local	$newnonce(x)$	variable $x \in V$	generating a nonce and storing in *x*
	$newskey(x)$	variable $x \in V$	generating & storing a symmetry key

Protocol Information. Protocol Information (denoted by *PI*) is defined during the specification extraction. A PI is a 5-tuple $(msg, ACSeq, ch, lc, BR)$, where $PI.msg$ is a concatenation of terms representing the message transmitted by the corresponding *TR*, and $PI.ACSeq = \langle AC_1, AC_2, ..., AC_\gamma \rangle$ is a sequence of action information where γ is the total number of actions. An action information AC_i is a 3-tuple (u, a, X) where $u (\in P)$ is the participant who performed the action, *a* is the name of action and *X* is a set of terms taken as parameters to *a*. $PI.ch$ is the communication channel. Further, if the message $PI.msg$ needs to be transmitted between two sub-components within a device, which acts on different protocols, the algorithm introduces local communication actions (e.g., between HS and ZFE of *hub* shown by the broken lines in Fig.1). $PI.lc \in P$ is the receiver ($lc \notin TR.R$) when local communication between two sub-components exists. $PI.BR$ is the branch information.

Parameterized Labeled Transition System. A traditional labeled transition system (LTS) is a 4-tuple $\mathcal{L} = (S, s_0, A, \rightarrow)$ where *S* is a set of states (locations); $s_0 \in S$ is the initial state; *A* is a set of actions; $\rightarrow \subseteq S \times A \times S$ is a labeled transition relation. We extend the LTS with parameters to differentiate the instances of the same behavior pattern to facilitate the attacker modeling. For example, we use the parameter $HubID'$ to represent the identity of the malicious hub compared with the *HubID* for the benign hub.

IV. SPECIFICATION EXTRACTION

The goal of specification extraction is to generate a representation of system integration. One challenge that can be foreseen is the gap between the execution traces (to be precise, the transactions after pre-processing) and the target LTS. To bridge the gap, we design a two-step extraction

```
input : $(TRSet, PS, EL, IK, TC)$
output: A List $PIL = [PI^1, PI^2, ..., PI^\delta]$ where $\delta = \beta$; Each
        transaction in $TRSet$ is mapped to a $PI$.
1  $F = \{f_1, f_2, ..., f_\eta\}$ where $\eta$ is the number of selected hash,
     cryptography and encoding/decoding functions.;
2  $g : GEVSet \times \mathbb{P}(TRSet)$ is relation indicates the transactions which a
     value appears.;
3  $TRSet_{new} \leftarrow TRSet$, $TRSet_{old} \leftarrow TRSet$;
4  do
5      $TRSet \leftarrow TRSet_{new}$;
6      $GEVSet = \bigcup EVSet_i (1 \leq i \leq \beta)$// global set of $EVSet$.;
7      $g \leftarrow Grouping(TRSet)$ ;
8      $GEVSet \leftarrow WB(GEVSet, PS, EL, IK)$;
9      $PIL \leftarrow Propagation(GEVSet, g)$;
10     $GEVSet \leftarrow ES(GEVSet, F, IK)$;
11     $PIL \leftarrow Propagation(GEVSet, g)$;
12     $GEVSet \leftarrow DA(GEVSet, PIL, TRSet, TRSet_{old}, IK)$;
13     $PIL \leftarrow Propagation(GEVSet, g)$;
14     $TRSet_{old} \leftarrow TRSet$;
15     $TRSet_{new} \leftarrow updateTRSet(TC.next)$ ;
16 while $TRSet_{new} \neq TRSet$;
17 return $PIL$;
```

Algorithm 1: PI Inference Algorithm

approach, which first extracts PIs from the transactions, and then transforms the PIs into LTS representations.

A. Inference of Protocol Information

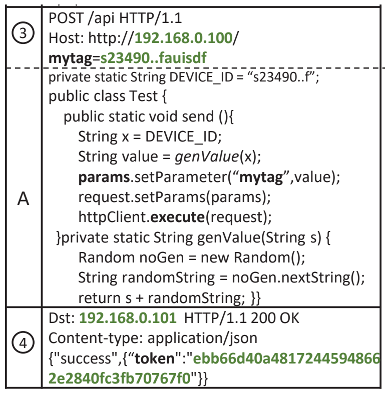
Given the transactions generated from trace processing, HOMESCAN uses several analysis techniques to infer the PIs using Algorithm 1. It takes a 5-tuple $(TRSet, PS, EL, IK, TC)$ as input, where $TRSet$ is the set of transactions; PS is the set of programs; EL is the set of execution logs; IK is the set of initial knowledge; TC is the set of test cases. The output of the algorithm is a list of inferred PI (PIL), each of which correlates with one transaction. The algorithm executes the next test case ($TC.next$ at line 15) and iteratively identifies new semantics until no new information can be found from the input resources. In each next iteration, the $TRSet_{new}$ includes new values and new branch information (BR) corresponding to the new configuration specified in the $TC.next$. HOMESCAN infers the types of new values using the techniques we detail in the remaining of this section.

Whitebox Analysis. HOMESCAN uses $WB(GEVSet, PS, EL, IK)$ (line 8) to infer each extracted value $EV.v$ of transaction TR. It identifies type information of a value $EV.v \in GEVSet$ and related local behaviors, and branch information by analyzing how v is generated or processed in the given program ($\in PS$).

First, HOMESCAN identifies the code snippet for analysis from the input PS. It uses the following two approaches for code snippet identification.

- If the source code is runnable, HOMESCAN logs API calls which send or receive the message of transaction TR (e.g., `execute()` of `DefaultHttpClient.class` with Java API of Apache) in the program.
- If only the executable files are available, HOMESCAN recovers the code using off-the-shelf reverse-engineering tools. Next, it scans the recovered code for existence of strings against a dictionary of strings. This dictionary is constructed with the extracted strings from the network trace corresponding to the message of TR. HOMESCAN confirms the code snippet by instrumenting the recovered code to obtain dynamic debug information, while executing the test case which triggers TR.

Next, HOMESCAN identifies associations among variables in the code snippet, for type information recognition of v. To this end, it uses data flow analysis techniques, including both backward and forward flow analysis. HOMESCAN assigns *Constant* terms to v if it is associated with a constant given as prior knowledge (e.g., IP address, cryptographic key) in the program. It assigns *Variable* terms if v is associated with a random value identified by the use of corresponding API (e.g., `java.util.Random`) calls. HOMESCAN assigns *Function* terms if v is associated with values generated by security sensitive functions (e.g., encryption, decryption and signature) which are identified by the cryptographic API (e.g., `javax.crypto.KeyGenerator`) calls. In addition, HOMESCAN records local actions related to the generation or processing of the value v. For example, if a new symmetric encryption key is generated, HOMESCAN records a local action $newskey(x)$ (listed in Table IV). For example, consider the message ③ in Fig. 4, which is transmitted by $TR_{③}$ in Fig. 2, and generated by the code snippet A listed in Fig. 4. The TR_3=$(3, CP, HS, \{(s23490..fauisdf, String, 0)\}, \varnothing)$ has one extracted value (EV_1), which is an HTTP request parameter. HOMESCAN infers $EV_1.v$ as terms `(DeviceID,nonce1)`. The corresponding PI^3 is (msg=`(DeviceID, nonce1)`, $ACSeq$=$\langle(CP, newnonce, \{nonce1\}), (CP, send, \{msg\}), (HS, recieve, \{msg\})\rangle$, ch=$wifi$, lc= $-$, BR=$\varnothing$).



```
③  POST /api HTTP/1.1
   Host: http://192.168.0.100/
   mytag=s23490..fauisdf
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   private static String DEVICE_ID = "s23490..f";
   public class Test {
      public static void send (){
          String x = DEVICE_ID;
          String value = genValue(x);
A         params.setParameter("mytag",value);
          request.setParams(params);
          httpClient.execute(request);
    }private static String genValue(String s) {
          Random noGen = new Random();
          String randomString = noGen.nextString();
          return s + randomString; }}
④  Dst: 192.168.0.101  HTTP/1.1 200 OK
   Content-type: application/json
   {"success",{"token":"ebb66d40a4817244594866
   2e2840fc3fb70767f0"}}
```

Fig. 4: Part of Captured HTTP Traces and Part of CP Source Code for the System in Fig.1

HOMESCAN identifies the branch information resulted from configuration changes in the smart home system. For example, different privileges may be assigned to different user (e.g., general and guest users) or CP (e.g., mobile or desktop app) configurations. With this approach, HOMESCAN uncovers control flows (e.g., if-else and case-switch) in a given program, and utilizes all input resources (e.g., mobile and desktop CP source code) during PI inference. To formalize the configurations, we assume the finite configuration set $\mathbb{C}$=$\{C^1, C^2, ..., C^i, ..., C^\lambda\}$ where λ is the number of configurations that can be changed (e.g., $\mathbb{C}$ =$\{C^{user}, C^{CP}\}$ where C^{user}=$\{general, guest\}$ and C^{CP}=$\{mobile, desktop\}$). As an example, in the LIFX system that we studied, the *desktop* app (CP) allows to control the SD over SD's open Wi-Fi hotspot whilst the *mobile* app forces to setup the SD with the home Wi-Fi before starting the control. Hence, HomeScan records the control (over open Wi-Fi) and setup (with home Wi-Fi) actions as two option-branches in the *PI* corresponding to the discovery success transaction.

Exhaustive Search. HomeScan uses exhaustive search to identify the type of a value with respect to a known function applied on a subset of extracted values. Hence, in this search, a finite set of existing functions are executed on all extracted values to check whether the values of unknown types can be generated. As shown in Algorithm 1, the *GEVSet* is input into the $ES(GEVSet, F, IK)$ with F a set of existing functions (e.g., MD5, SHA-1 and Base64) and *IK*. For example, consider v="ebb66d...0767f0" in our running example (④ in Fig. 4). HomeScan performs all the existing hash functions on the values it has collected in *GEVSet*. Once it finds that $SHA1$(`nonce1`) has the same value, it can infer that the type of this value is a hash value over `nonce1`.

Differential Analysis. HomeScan uses $DA(GEVSet, PIL, TRSet, TRSet_{old}, IK)$ to infer the types based on the associations from two categories of changes, i.e., configurations and control commands. HomeScan identifies the association for the difference of the v in $TRSet_{old}$ and $TRSet$ for the value with identity $EV.id \in TR$. Further, HomeScan triggers the trace capturing component to re-execute a particular test case during an analysis to assure the consistency of values $EVSet \in TR$.

Configuration Changes. In our generic architecture, the configuration $\mathbb{C}$=$\{C^{hub}, C^{SD}, C^{CP}\}$ is a set of participants. Hence, for example, HomeScan can substitute the hub with other hubs using the same interface (e.g., the communication protocol), i.e., C^{hub}=$\{hub_1, hub_2, ..., hub_\mu\}$ where μ indicates the number of the hubs under the control of HomeScan, to check the difference of the target $EV.v$ against the change of the hub. For a value $EV.v$ whose domain-specific type is unknown, HomeScan infers its type (t) as follows.

- If C^i and $EV.v$ always change together, then they are likely correlated, e.g., `HubID` in the running example.
- If $EV.v$ always changes in every execution, then it is likely a session-specific random nonce, e.g., `nonce`.
- If $EV.v$ keeps constant, then it is likely a protocol-specific value, e.g., `UPnPMsearchRequest`.

Control Command Changes. During the control stage, the commands sent to the SD may be encrypted. HomeScan exploits the association between the control commands and the meta-data of the encrypted messages by using differential analysis, to infer the types (e.g., ON/OFF/color-change command) of the encrypted messages. According to the connection through which a control command can be sent to the SD, HomeScan uses the following approaches to infer its type.

- ***Persistent Connection.*** Typically, the heartbeats are required in order to maintain a persistent connection. In this scenario, the packets including the commands may be inundated by the heartbeat packets. To remove the packets of the heartbeat from the trace, HomeScan captures the packets when *no command* is issued by the CP, and labels it as the heartbeat. This enables HomeScan to remove the heartbeat packets from the trace and infers the remaining packets as the control command(s).
- ***Non-persistent Connection.*** In non-persistent connection, a handshake is often used to establish the connection before a control command is sent. Therefore, given a trace of control command execution, HomeScan identifies the packets on the trace corresponding to three different stages in a handshake based protocol (⟨*connection, command, disconnection*⟩). To achieve this, HomeScan reruns test cases for *different* control commands. The packets common in all runs are considered to be relevant to *connection* and *disconnection* stages. The remaining packets are inferred as the *command* data packets.

```
input : $PIL$
output: A List $LTSL = [LTS_1, LTS_2, ..., LTS_\sigma]$ where $\sigma = \mid P \mid$.
        Each participant in $P$ is mapped to an $LTS$
for $p \in P$ do
    $src_p \leftarrow s_0$, $dst_p \leftarrow null$, $LTS_p = (src_p, \{src_p\}, \varnothing, \varnothing)$;
    foreach $PI^q \in PIL$ do $s_p^q \leftarrow null$;
end
for $PI^q \in PIL$ do
    $PI^q.ACSeq \frown CreateLCActions(PI^q.msg,\ PI^q.lc)$;
    $uch \leftarrow UniqueCH(PI^q.ch)$;
    for $ac \in PI^q.ACSeq$ do
        $p = ac.u$, $l \leftarrow CreateLabel(ac, uch)$;
        $dst_p \leftarrow GenState(ac)$;
        if $(s_p^q \neq null)$ then $src_p \leftarrow s_p^q$;
        $LTS_p.A \leftarrow LTS_p.A \vee \{l\}$, $LTS_p.S \leftarrow LTS_p.S \vee \{dst_p\}$;
        $LTS_p.Tr \leftarrow LTS_p.Tr \vee \{src_p, l, dst_p\}$;
        $src_p \leftarrow dst_p$, $s_p^q \leftarrow null$;
        for $TR.id \in BR$ do
            if $(q < TR.id)$ then $s_p^{TR.id} \leftarrow dst_p$;
            else if $(q = TR.id)$ then
                $LTS_p.Tr \leftarrow LTS_p.Tr \vee \{dst_p, l, dst_p\}$;
            else if $(q > TR.id)$ then
                $exdst_p \leftarrow GenState(AC_1 \in PI^{TR.id}.ACSeq)$;
                $LTS_p.Tr \leftarrow LTS_p.Tr \vee \{dst_p, l, exdst_p\}$;
        end
        $LTSL \leftarrow LTS_p$
    end
end
return $LTSL$;
```

Algorithm 2: LTS Representation Algorithm

Type Propagation. HomeScan uses $Propagation(GEVSet, g)$ to propagate the type of a particular value to its other occurrences. The rationale is that in security protocols, a data item typically appears in multiple steps (as shown with red arrows in Fig. 2). HomeScan exploits this feature to propagate inferred information in different transactions. To do the propagation, HomeScan generates groups *GEVSet* (line 4 in Algorithm 1) to track all occurrences of each value throughout all transactions. Later after each analysis, the updated *GEVSet* and the groups are input to propagation component for the *PI* generation.

B. LTS Generation

After extracting the PIs, HomeScan translates them into the LTS representations. Algorithm 2 shows our approach. It takes the *PIL* (output of Algorithm 1) as input and generates a list of LTSs. It begins with initializing an LTS_p for each participant $p \in P$ with the initial state (s_0), the set of states

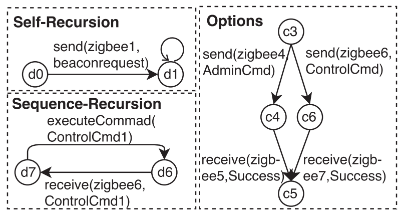
(S), the set of actions (A), and the set of transitions (Tr) in a tuple $(s_0, \{s_0\}, \varnothing, \varnothing)$ (lines 1-4). Then it iterates through the *PIL* and transforms each *PI* into *LTS* transitions. First, it extends the *PI.ACSeq*, if a private communication exists (line 6). Next, it creates a unique channel (line 7) before creating an action label (line 9). Once the source and destination states and labels are created (lines 9-11), it updates the *LTS* components of participant p identified at line 9. If the *PI* has branch information, it either records the source state of options (line 16), adds self-recursions (line 18), adds sequence-recursions, or merges branches (lines 20-21). Below, we detail the LTS generation.

States. A transition involves two states. Its source state is denoted by src_p, while the destination state is denoted by dst_p. In addition, HOMESCAN uses state s_p^q to track the src_p of a branch, where q is the transaction ID (*TR.id*). The dst_p is given by the function *GenState* (line 10). If the input *ac* represents a new action, *GenState* outputs a new dst_p. If the action has been mapped to a dst_p by the function before, the function outputs the existing dst_p. Moreover, the src_p of the immediate transition is the dst_p of the current transition, when it is not a branch (line 14).

Actions and Transitions. During the iterations through *PIL*, the information in each PI^q is used to create labels (actions). The $PI^q.ACSeq$ states the action information with their sequence. The algorithm creates labels for actions in the stated order (e.g., $\langle AC_1, AC_2, AC_3, AC_4\rangle$ where AC_1 and AC_2 are local actions conducted by the sender, $AC_3 = (se, send, msg)$ is an action of message sending, and $AC_4 = (r_i \in R, receive, msg)$ is an action of message receiving). Further, HOMESCAN uses the *CreateLCActions* function to add information of the local sending and local receiving actions to $PI^q.ACSeq$ (e.g., $PI^q.ACSeq^\frown\langle(r_i \in R, send, msg), (lc, receive, msg)\rangle$ (line 6).

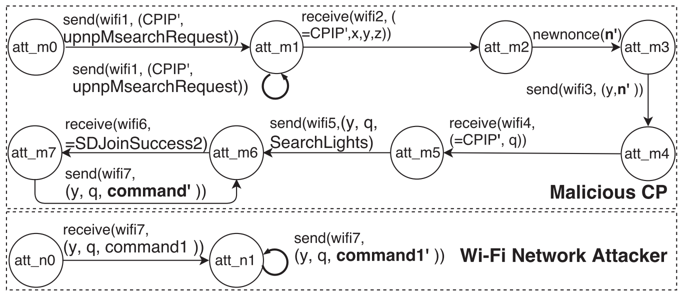
Each label is created using the function *CreateLabel* (line 9). The input to the function, i.e., *ac*, has information about action (a and X). If *ac* is a local action, then $a \in \{newnonce, newkey, newkeypair, executeCommand\}$ and $X \in T$. If *ac* is a communication action, then $a \in \{send, receive\}$ and $X = msg$. The input *uch* generated using the *UniqueCH* function is used to send/receive the *msg* via a unique channel (line 7). If *ac* is a local communication, then the *CreateLabel* function uses a unique private channel to transmit the *msg*. Once the label and the next state are ready, LTS_p is updated such that $src_p \xrightarrow{l} dst_p$ is added (lines 12-13).

Branches. If the PI^q includes information about branches (represented by $TR.id \in BR$), it is analyzed from line 15 to line 22. Fig. 5 shows different types of branches in an LTS. If *TR.id* of the branch is greater than that of the current *PI*, it is an *option*. Hence, current dst_p is tracked using $s_p^{TR.id}$ (line 16). After it is set, $s_p^{TR.id}$ is taken as the src_p (line 11) in the next iteration. If the *TR.id* of the branch is the same as that of *PI*, this branch is a *self-recursion*. It is represented as an edge from dst_p to dst_p (line 18). Otherwise, the $PI^{TR.id}$ is already processed. Hence, the dst_p of the first action (as stated in sequence $PI^{TR.id}.ACSeq$) of the branch exists. The *GenState* function returns that existing state as $exdst_p$. HOMESCAN adds a transition from the current state dst_p to $exdst_p$ (lines 20-21). This is called a branch merge. If the first action of the branch exists in the current path (root to the src_p), this branch is a *sequence-recursion*. Hence, HOMESCAN merges the current and existing src_p states. After all actions are processed, the LTS representation is generated.


Self-Recursion
send(zigbee1, beaconrequest)
d0
d1
Sequence-Recursion
executeCommad(ControlCmd1)
d7
d6
receive(zigbee6, ControlCmd1)
Options
c3
send(zigbee4, AdminCmd)
send(zigbee6, ControlCmd)
c4
c6
receive(zigb-ee5,Success)
receive(zigb-ee7,Success)
c5
Fig. 5: Types of Branches in an LTS

V. FLAW IDENTIFICATION

After the specification extraction, the LTS representation is generated to model the behaviors of the participants and their communications. We can further analyze the security properties of the extracted protocol by verifying the generated LTS model against the attack models.


att_m0
send(wifi1, (CPIP', upnpMsearchRequest))
att_m1
send(wifi1, (CPIP', upnpMsearchRequest))
receive(wifi2, (=CPIP',x,y,z))
att_m2
newnonce(n')
att_m3
send(wifi3, (y,n'))
att_m4
receive(wifi4, (=CPIP', q))
att_m5
send(wifi5,(y, q, SearchLights)
att_m6
receive(wifi6, =SDJoinSuccess2)
att_m7
send(wifi7, (y, q, command'))
Malicious CP
att_n0
receive(wifi7, (y, q, command1))
att_n1
send(wifi7, (y, q, command1'))
Wi-Fi Network Attacker
Fig. 6: LTS Representation for the Malicious CP and Wi-Fi Attacker

In HOMESCAN, the behavior of an attacker is modeled as an LTS $\mathcal{L}_{att} = (S, s_0, A_{att}, \rightarrow_{att})$, where A_{att} is a set of actions performed by the attacker. In Fig. 6, we illustrate the behaviors of the malicious participants and the network attacker using the examples of the malicious CP and the Wi-Fi network attacker in the running example. Note that we assume the attacker is able to check and receive a specific message which is modeled as a guard on the received messages (denoted by "="). The malicious CP pretends to be an honest one in the same network with a different IP address ***CPIP'***. It sends out its newly generated nonce n' (state *att_m*2), trying to receive an authenticated user ID $\mathbf{hash}(n')$ (we store this value in a variable q in the LTS in Fig. 6) from the hub (state *att_m*4). Once successful, the malicious CP is able to find the SDs within the network it has joined and then controls them by sending its own malicious command ***command'*** (state *att_m*7). The Wi-Fi network attacker resides between the CP and the HS. It is able to intercept and replace the command sent from the honest CP with ***command1'*** (state *att_n*1).

Given the extracted LTS models of both participants and attackers, HOMESCAN generates the execution of the whole smart home system defined in Definition 1.

Definition 1 (System LTS Generation) *Let* $\mathcal{L}_i = (S_i, s_{0_i}, A, \rightarrow_i)$ *be the model of participant i,* $\mathcal{L}_{att} = (S_{att}, s_{0_{att}}, A_{att}, \rightarrow_{att})$ *be the attack model,* NS_{att} *be the attacker's knowledge set,* *and A_c be the set of sending A_s and receiving actions A_r ($A_s, A_r \subseteq A_c \subseteq A$). The model of the whole system is an LTS $(S, s_0, A', \rightarrow)$, where $S \subseteq S_1 \times \cdots S_n \times (S_{att} \times \mathbb{P}T)$, initial state $s_0 = (s_{0_1}, \cdots, s_{0_n}, (s_{0_{att}}, \varnothing))$, $A' = A \cup A_{att} \cup A_{sr}$, $A_{sr} = (A_s \times A_r)$ is a set of sending and receiving action pairs denoting synchronization, and $\rightarrow \subseteq S \times A' \times S$ is the transition relation.*

Due to the page limitation, we list part of our LTS generation rules in Fig. 7, and the full list can be found in our technical report [6]. Here we intuitively introduce it. Rule *comm* denotes a communication action between two honest participants. Rules *att_rec* and *att_send* represent the attacker's capabilities. *att_rec* captures the message sent from an honest participant and those generated by the attacker (attacker can apply a cryptographic function to the captured message and generate new terms using function *Upd*). These new terms are added to the set NS_{att}. *att_send* sends out a fake simulated message to pretend as an honest participant. Rule *att_send_all* represents the network attacker's capability that it can intercept the communication between honest participants and thereafter randomly send a message from its knowledge set NS_{att} to the intercepted honest receiver.

Notice that we define an additional sending action $send(ch, \forall)$ to represent the network attacker's capability of sending any message from the attacker's knowledge set $NS_{att} \subset K$ where the knowledge set K is a set of terms. According to Definition 2, an attacker has the capability of updating his knowledge set NS_{att} by applying the attacker knowledge's set update function *Upd* defined as follows.

Definition 2 (Attacker Knowledge Set Update) *Let NS_{att} and NS'_{att} be the input and output of the attacker's knowledge update function Upd such that $NS'_{att} \leftarrow Upd(NS_{att})$. Let m, n, $pk, sk \in T$ where pk and sk represent a public-private key pair such that:*

$$NS'_{att} \leftarrow NS_{att} \cup \begin{cases} \{\boldsymbol{senc}(m,n)\}, & m, n \in NS_{att} \\ \{m\}, & \boldsymbol{senc}(m,n), n \in NS_{att} \\ \{\boldsymbol{aenc}(m,pk)\}, & m, pk \in NS_{att} \\ \{m\}, & \boldsymbol{aenc}(m,pk), sk \in NS_{att} \\ \{\boldsymbol{sign}(m,sk)\}, & m, sk \in NS_{att} \\ \{m\}, & \boldsymbol{sign}(m,sk), pk \in NS_{att} \\ \{\boldsymbol{hash}(m)\}, & m \in NS_{att} \end{cases}$$

In order to verify the security properties, HOMESCAN applies the reachability analysis to the generated execution of the smart home systems, using the classical algorithms such as BFS and DFS. It determines whether a vulnerability exists by searching whether a particular state (referred to *bad state* hereinafter) can be reached in the whole system. For example, in order to determine if the CP can have unauthorized control of the hub and the SD, we can query if the system execution in the running example can reach state *att_m*6 from state *att_m*7 in Fig. 6. Alternatively, we can also query the existence of a particular set of terms in the attacker's knowledge set NS_{att} to determine if the attacker has enough information to launch an attack. For example, we can query if the set $\{command', \mathbf{hash}(n')\}$ exists in the attacker's knowledge set in Fig. 6 to determine if the malicious CP can have unauthorized control of the hub and the SD.

$$\frac{s_i \xrightarrow{Send(ch,M)}_i s'_i,\ s_j \xrightarrow{Receive(ch,x)}_j s'_j,\ a_i = Send(ch,M),\ a_j = Receive(ch,x)}{(s_1, \cdots, s_i, \cdots, s_j, \cdots s_n, (s_{att}, NS_{att})) \xrightarrow{(a_i, a_j[M/x])} (s_1, \cdots, s'_i, \cdots, s'_j, \cdots s_n, (s_{att}, NS_{att}))}\ [\ comm\ ]$$

$$\frac{s_i \xrightarrow{Send(ch,M)}_i s'_i,\ s_{att} \xrightarrow{Receive(ch,x)}_{att} s'_{att},\ a_i = Send(ch,M),\ a_{att} = Receive(ch,x)}{(s_1, \cdots, s_i, \cdots, s_n, (s_{att}, NS_{att})) \xrightarrow{(a_i, a_{att}[M/x])} (s_1, \cdots, s'_i, \cdots, s_n, (s'_{att}, Upd(NS_{att} \cup \{M\})))}\ [\ att\_rec\ ]$$

$$\frac{s_i \xrightarrow{Receive(ch,x)}_i s'_i,\ s_{att} \xrightarrow{Send(ch,M)}_{att} s'_{att},\ a_i = Receive(ch,x),\ a_{att} = Send(ch,M)}{(s_1, \cdots, s_i, \cdots, s_n, (s_{att}, NS_{att})) \xrightarrow{(a_{att}, a_i[M/x])} (s_1, \cdots, s'_i, \cdots, s_n, (s'_{att}, NS_{att}))}\ [\ att\_send\ ]$$

$$\frac{s_i \xrightarrow{Receive(ch,x)}_i s'_i,\ s_{att} \xrightarrow{Send(ch,\forall)}_{att} s'_{att},\ a_i = Receive(ch,x),\ \exists M_i \in NS_{att} \bullet a_{att} = Send(ch,M_i)}{(s_1, \cdots, s_i, \cdots, s_n, (s_{att}, NS_{att})) \xrightarrow{(a_{att}, a_i[M_i/x])} \rightarrow (s_1, \cdots, s'_i, \cdots, s_n, (s'_{att}, NS_{att}))}\ [\ att\_send\_all\ ]$$

Fig. 7: Execution Rules where $x, x^{-1} \in V, M \in T$ and $ch \in C$

VI. EVALUATION

In this section, we present our experiment setup and overall results. The supporting materials are published online [6].

A. Subjects of Our Evaluation

Philips Hue. Philips Hue is a smart lighting system produced by Philips and is claimed to be the world's most popular smart lighting system [16]. The components and the working process of this system are similar to the example discussed in Section II-A. The detailed description is available in the technical report [6].

LIFX. LIFX smart lighting system comprises a CP and a SD (i.e., the smart bulb). The SD is Wi-Fi enabled and initially provides an open Wi-Fi hotspot. In the discovery stage, the CP joins the SD's hotspot and discovers the configurations of the SD by broadcasting a UDP packet. In the authentication stage, the CP sends credentials of the home Wi-Fi to the SD over SD's hotspot. With the received credentials, the SD joins the home Wi-Fi. In the control stage, the SD can be controlled by any CP which joins the home Wi-Fi.

Chromecast. Google's Chromecast allows streaming a video to a TV. It comprises a CP, a receiver SD, and a Google's server (denoted by GS). The Chromecast SD also provides an open Wi-Fi hotspot. In the discovery stage, the CP joins SD's hotspot and receives the device information (e.g., `PublicKey`) of the SD. In the authentication stage, the CP sends the credentials of the home Wi-Fi to the SD over SD's hotspot. Once the SD has joined the home Wi-Fi, the CP uses Multicast DNS (MDNS) to discover the SD. Further, to pair the CP and the GS, the CP sends the `ScreenID` of the SD to the GS. The GS responds to the CP with a `token`. In the control stage, the CP uses this `token` to authenticate its YouTube-video-casting requests with the GS.

B. Setup

Trace Capturing and Pre-Processing. We use 2.4 GHz deRFusb23-E00 USB sniffing radio stick and Perytons Analyzer to capture ZigBee traces, and Wireshark tool to capture the Wi-Fi traffic. We use Xposed framework [17] to obtain the execution log of the Android app (i.e., the CP).

TABLE V: Summary of the Vulnerabilities

	Misresponse to Discovery Request	Flawed Authentication Protocol	Lack of Authorization	Use of Insecure Underlying Protocols	Unprotected SD's Wi-Fi Hotspot	Lack of Device or User Authentication Protocol	Vulnerable to Network Traffic Replay
Philips Hue	2	1	1	1	0	0	0
LIFX	0	0	0	0	2	1	1
Chromecast	1	0	0	0	1	1	0

PI Inference and LTS Representation. The detailed LTSs for the three systems are available in the technical report [6].
Flaw Identification. HOMESCAN uses a model checker called PAT [18] as the inference engine in these case studies. By analyzing the LTS representations of the systems against the attack models defined in Section II-B, HOMESCAN reports twelve security flaws shown in Table V.

C. Results

HOMESCAN discovered seven categories of vulnerabilities.
Mis-response to Discovery Request. During the discovery stage, participants send/receive discovery requests to identify potential participants of the system. However, if a participant fails to validate the source of discovery requests, it may incorrectly respond to the attacker. HOMESCAN identifies three vulnerabilities which belong to this category. First, Philips Hue HS replies to discovery requests, from any UPnP (a known flawed protocol [19]) enabled devices. Second, Philips Hue ZFE always replies to the discovery requests from ZigBee enabled devices. Third, the Chromecast SD replies to MDNS discovery requests from any device in the home Wi-Fi.
Flawed Authentication Protocol. Due to the resource limitations, smart home systems may adopt customized authentication protocols. This may result in flawed protocols. HOMESCAN identifies one vulnerability from Philip Hue which can be exploited by a malicious CP. In the authentication stage, the Philips Hue HS relies on the user to press the button on the *hub* to enable the authentication token generation. However, after the pressing, this protocol does not guarantee that the HS only generates the token to the benign CP requests.
Lack of Control to Administration Commands. In the control stage, the CP is allowed to send administration commands, such as adding/removing SDs. However, this permission should be limited to authorized parties. HOMESCAN identifies one vulnerability from Philips Hue—any CP authenticated by the HS, instead of only the admin user, can re-configure Philips Hue. This may lead to severe consequences, including uncontrolled authentication and denial-of-service against both the *hub* and the SD.
Use of Insecure Underlying Protocols. Smart home systems typically rely on existing protocols, but some of them may select an insecure one. HOMESCAN identifies such a vulnerability from Philips Hue, which uses ZLL for authentication. However, ZLL is designed to allow a participant to reset the established connection. In particular, after the SD and the hub have established a connection though ZLL, the attacker can send a `NextworkJoinRequest` to the SD to trigger it to re-execute the protocol. After that, the attacker can impersonate as a hub to establish another connection with the SD.
Unprotected SD's Wi-Fi Hotspot. SDs may come with on-board open Wi-Fi hotspots. These unprotected Wi-Fi hotspots can be exploited by malicious participants at all stages of the system. HOMESCAN identifies three vulnerabilities which belong to this category. First, in the discovery stage of LIFX, any CP which joins the SD's hotspot can obtain the SD's configurations and forcefully connect the SD to an attacker's Wi-Fi. Another vulnerability of this category are found in the CPs of the LIFX and Chromecast, which cause them to be deceitfully connected to a fake SD's hotspot. This vulnerability leads to a severe consequence in LIFX's authentication stage, where the CP sends the credentials of the home Wi-Fi in plain text so that the attacker can exploit this vulnerability to steal these credentials.
Lack of Device or User Authentication Protocol. Due to the resource limitations, smart home systems may be developed without any authentication protocol. These systems can be exploited by malicious participants to take over control or obtain sensitive information. HOMESCAN identifies two vulnerabilities of this category. In the LIFX system, any CP which joins the home Wi-Fi can control the SD. Similarly, but with a serious consequence, a malicious CP in the Chromecast system which joins the home Wi-Fi can obtain the `VideoID` of a private YouTube video and cast it to the TV screen.
Vulnerable to Network Traffic Replay. The network packets exchanged among participants over channels may not include any session related data (e.g., timestamp and nonce). These packets can be intercepted and later replayed by a network attacker who taps on the communication channel. HOMESCAN identifies one vulnerability which belong to this category. The UDP packets sent by LIFX CP can be intercepted and replayed by a network attacker to manipulate the victim SD.

VII. RELATED WORK

A. Specification Extraction

There exists different approaches [20], [21], [22], [23], [24] to build the model of a system through static code analysis or execution trace analysis. However, the existing approaches are not directly applicable to the specification extraction of a smart home systems, due to the challenge of partial availability of the implementation. Hence, as a compliment to existing work, we propose a hybrid approach in this work.

B. IoT Security

The research of IoT security mainly focuses on three domains, i.e., IoT devices, protocols and platforms.
Security of IoT Devices. Ho et al. [25] present flaws in the design of smart locks and show how they lead to unauthorized home access. Fawaz et al. [26] propose a system that protects

BLE equipped devices from privacy leakages during the device discovery. Das et al. [27] have discovered privacy leakage in BLE network traffic of wearable fitness trackers.

Security of IoT Protocols. Ronen et al. [28] discover a worm attack against Philips Hue lamps by exploiting the ZigBee protocol. Zilliner et al. [29] show that the actual implementations of ZigBee certified smart devices have insufficient security controls. Santos et al. [30] reveal the information leakage on ZigBee network and propose countermeasures. Armknecht et al. [31] discuss attacks on the ZLL by studying its specifications. Fouladi et al. [32] demonstrate that proprietary Z-Wave protocol vulnerabilities could lead to remote unlocking of locks. Siby et al. [33] propose IoTScanner which provides an overview of operations in all observed wireless networks. Choi et al. [34] develop an automatic spoofer tool which reconstructs protocols over IEEE 802.15.4. Compared with these studies, our work focuses more on the application layer of the integration of such protocols which may introduce novel attacks.

Security of IoT Platforms. Jia et al. [4] propose a context-based permission system for applied IoT platforms. Fernandes et al. [35] propose an approach to address how the sensitive data processed by third party apps after obtaining the access. Fernandes et al. [12] demonstrate that CP applications could be exploited by evaluating the security design of Samsung SmartThings framework. The existing studies mainly focus on the application frameworks, which is part of our consideration in our work.

VIII. Conclusion and Future Work

We present HomeScan, a semi-automatic approach to extract the abstract specification of the application-layer protocol and internal behaviors of smart home systems from their implementations, whereby it is possible to conduct an end-to-end security analysis against various practical attack models. Using HomeScan, we have found twelve security vulnerabilities from three real-world smart home systems. Our work has demonstrated the necessity of considering the security issues in IoT systems from the perspective of integration.

References

[1] Y. Oren and A. D. Keromytis, "From the Aether to the Ethernet-Attacking the Internet using Broadcast Digital Television," in *USENIX Security*, 2014, pp. 353–368.

[2] K. Townsend, "Attacking smart TVs ," http://itsecurity.co.uk/2014/06/attacking-smart-tvs/, 2017.

[3] Y. Michalevsky, S. Nath, and J. Liu, "Mashable: mobile applications of secret handshakes over bluetooth le," in *MobiCom*, 2016, pp. 387–400.

[4] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "Contexiot: Towards providing contextual integrity to appified iot platforms," in *NDSS*, 2017.

[5] R. M. Keller, "Formal verification of parallel programs," *Communications of the ACM*, vol. 19, pp. 371–384, 1976.

[6] HomeScan. https://sites.google.com/view/homescandemo/home.

[7] Samsung SmartThings. http://www.samsung.com/us/smart-home/.

[8] HomeGenie. https://genielabs.github.io/HomeGenie/.

[9] "Zigbee light link standard version 1.0," http://www.newsroom.lighting.philips.com/news/2017/20170831-philips-hue-marks-5th-birthday-with-new-products-and-entertainment-capability, 2012.

[10] M. M. Hossain, M. Fotouhi, and R. Hasan, "Towards an analysis of security issues, challenges, and open problems in the internet of things," in *IEEE SERVICES*, 2015, pp. 21–28.

[11] T. Denning, T. Kohno, and H. M. Levy, "Computer security and the modern home," *Communications of the ACM*, vol. 56, pp. 94–103, 2013.

[12] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE S&P*, 2016, pp. 636–654.

[13] H. Ryu and J. Kwak, "Secure data access control scheme for smart home," in *Ubicomp*, 2015, pp. 483–488.

[14] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in internet of things: The road ahead," *Computer Networks*, pp. 146 – 164, 2015.

[15] O. Mouaatamid, M. Lahmer, and M. Belkasmi, "Internet of things security: Layered classification of attacks and possible countermeasures," *Electronic Journal of Information Technology*, 2016.

[16] P. den Dunnen. Philips. http://www.newsroom.lighting.philips.com/news/2017/20170831-philips-hue-marks-5th-birthday-with-new-products-and-entertainment-capability.

[17] Xposed. http://repo.xposed.info/.

[18] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "Pat: Towards flexible verification under fairness," in *CAV*, 2009, pp. 709–714.

[19] H. Moore, "Security flaws in universal plug and play: Unplug. dont play," https://hdm.io/writing/SecurityFlawsUPnP.pdf.

[20] T. D. B. Le and D. Lo, "Deep specification mining," in *Proceedings of the 27th ACM SIGSOFT ISSTA*. ACM, 2018, pp. 106–117.

[21] L. Mariani, M. Pezzè, and M. Santoro, "Gk-tail+ an efficient approach to learn software models," *IEEE TSE*, vol. 43, no. 8, pp. 715–738, 2017.

[22] T.-D. B. Le, X.-B. D. Le, D. Lo, and I. Beschastnikh, "Synergizing specification miners through model fissions and fusions (t)," in *IEEE ASE*, 2015, pp. 115–125.

[23] G. Bai, J. Lei, G. Meng, S. S. Venkatraman, P. Saxena, J. Sun, Y. Liu, and J. S. Dong, "Authscan: Automatic extraction of web authentication protocols from implementations." in *NDSS*, 2013.

[24] Q. Ye, G. Bai, K. Wang, and J. S. Dong, "Formal analysis of a single sign-on protocol implementation for android," in *ICECCS*, 2015, pp. 90–99.

[25] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart locks: Lessons for securing commodity internet of things devices," in *ASIACCS*, 2016, pp. 461–472.

[26] K. Fawaz, K.-H. Kim, and K. G. Shin, "Protecting privacy of ble device users," in *USENIX Security*, 2016, pp. 1205–1221.

[27] A. K. Das, P. H. Pathak, C.-N. Chuah, and P. Mohapatra, "Uncovering privacy leakage in ble network traffic of wearable fitness trackers," in *HotMobile*, 2016, pp. 99–104.

[28] E. Ronen, A. Shamir, A.-O. Weingarten, and C. OFlynn, "Iot goes nuclear: Creating a zigbee chain reaction," in *IEEE S&P*, 2017, pp. 195–212.

[29] T. Zillner and S. Strobl, "Zigbee exploited: The good the bad and the ugly," in *Black Hat*, 2015.

[30] J. Dos Santos, C. Hennebert, and C. Lauradoux, "Preserving privacy in secured zigbee wireless sensor networks," in *WF-IoT*, 2015, pp. 715–720.

[31] F. Armknecht, Z. Benenson, P. Morgner, and C. Müller, "On the security of the zigbee light link touchlink commissioning procedure," in *Gesellschaft für Informatik eV (GI)*, 2016, p. 229.

[32] B. Fouladi and S. Ghanoun, "Honey, i'm home !!-hacking z-wave home automation systems," in *Black Hat*, 2013.

[33] S. Siby, R. R. Maiti, and N. O. Tippenhauer, "Iotscanner: Detecting privacy threats in iot neighborhoods," in *IoTPTS*, 2017, pp. 23–30.

[34] K. Choi, Y. Son, J. Noh, H. Shin, J. Choi, and Y. Kim, "Dissecting customized protocols: Automatic analysis for customized protocols based on ieee 802.15.4," in *ACM WiSec*, 2016, pp. 183–193.

[35] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks," in *USENIX Security*, 2016, pp. 531–548.