FORESEE: A Cross-Layer Vulnerability Detection Framework for the Internet of Things

Zheng Fang*, Hao Fu§, Tianbo Gu*, Zhiyun Qian¶, Trent Jaeger‖, and Prasant Mohapatra*
* Department of Computer Science, University of California, Davis
§ DiDi Labs, CA, USA
¶ Department of Computer Science and Engineering, University of California, Riverside
‖ Department of Computer Science and Engineering, Pennsylvania State University
Email: {zkfang,tbgu,pmohapatra}@ucdavis.edu, haofu@didiglobal.com, zhiyunq@cs.ucr.edu, tjaeger@cse.psu.edu

***Abstract*—The exponential growth of Internet-of-Things (IoT) devices not only brings convenience but also poses numerous challenging safety and security issues. IoT devices are distributed, highly heterogeneous, and more importantly, directly interact with the physical environment. In IoT systems, the bugs in device firmware, the defects in network protocols, and the design flaws in system configurations all may lead to catastrophic accidents, causing severe threats to people's lives and properties. The challenge gets even more escalated as the possible attacks may be chained together in a long sequence across multiple layers, rendering the current vulnerability analysis inapplicable. In this paper, we present FORESEE , a cross-layer formal framework to comprehensively unveil the vulnerabilities in IoT systems. FORESEE generates a novel attack graph that depicts all of the essential components in IoT, from low-level physical surroundings to high-level decision-making processes. The corresponding graph-based analysis then enables FORESEE to precisely capture potential attack paths. An optimization algorithm is further introduced to reduce the computational complexity of our analysis. The illustrative case studies show that our multilayer modeling can capture threats ignored by the previous approaches.**

I. INTRODUCTION

With the rapid development of IoT technologies, billions of IoT devices are deployed in the world. Various communication protocols, applications, and platforms are designed for diverse application scenarios. Popular IoT platforms, such as Samsung SmartThings, Apple HomeKit, etc., attract more and more developers to design numerous applications to automate our lives. For example, there are more than 5,000 active developers and 75 million Applets since the launch of IFTTT platform [1]. According to a report from the McKinsey Global Institute, it is estimated that the IoT could have an annual economic impact of $3.9 trillion to $11.1 trillion by 2025 [2].

Despite numerous benefits that IoT provides us, it has also brought tremendous challenges to safety and security analysis. The heterogeneity of IoT, such as diverse applications and protocols, makes it challenging to design a general solution to resolve all the safety and security issues. Most existing research only tackles the security issues of one single component in IoT. For example, [3] investigates the Mirai botnet of IoT devices, which is caused by the defects of communication protocols. [4, 5] focus on the security and privacy of the applications. HoMonit [6] discovers the anomaly behaviors of applications via analyzing the corresponding wireless traffic, which only considers communication and application components. Because these works focus on the security issues of one or two components, they cannot discover other potential threats in other components of the IoT system. For instance, the physical environment and users' behaviors, chained by IoT devices and applications, may also pose unknown threats to IoT systems. The heterogeneity and interdependent components of IoT require a more complicated detection framework that considers all the components simultaneously.

To solve these challenges, we propose a cross-layer vulnerability analysis framework named FORESEE to detect the vulnerabilities in the IoT system. Unlike existing work that generally focuses on certain malicious activities at a single component, we target all the IoT components which can trigger security and privacy problems. We first decouple and model the IoT components in a layered structure, which consists of *physical environment layer*, *device layer*, *communication layer* and *application layer*. The layered structure not only includes more components of the system but also considers the interaction between them. Furthermore, we show how to decompose real-world attacks and integrate them into the multilayer graph, which enables us to analyze how they propagate and undermine system security.

The benefits of our approach are threefold. First of all, by considering all of the core components simultaneously, we can discover more vulnerabilities than the existing frameworks. For example, suppose the user is watching TV at home, and then leaves home without turning off the TV. If the show on TV plays human voice "open the door", the voice assistant may sense the command and issue a door-open command. This example shows that user's behavior can affect system security and thus should be included in the analysis framework. Second, we thoroughly analyze how different exploits cooperate and breach the system. For instance, if an air conditioner is plugged into a smart outlet and the outlet has weak default credential, then the attacker can remotely log into the outlet and disable the air conditioner, even triggering other potential actions such as opening the window, which may cause severe security issues. Frameworks focusing solely on software applications cannot discover such vulnerabilities because they involve physical dependence between devices and brute-force login at the communication layer. Lastly, we can examine how

seemingly relatively unimportant vulnerabilities can escalate and cause disastrous results due to the interconnected nature of IoT devices. This helps us better evaluate the vulnerabilities' impact on system security and prioritize the fixing of them.

Based on the idea of multiple layers, we model system states at these layers and formalize the mapping between states at adjacent layers, resulting in a multilayer state transition graph. Moreover, we explore all the existing and potential attacks and incorporate these attacks into the graph to form a final *hypothesis graph*. Then we apply model checking technique to detect various vulnerabilities and attacks. With the rapid increase of the devices and applications, the number of nodes (which correspond to system states) explodes, leading to massive computation cost. Therefore, we design a state compression algorithm to intelligently generate independent sub-graphs for vulnerability detection.

In summary, we make the following contributions:

- We formally define IoT systems using a hierarchical model, which precisely reflects the flows of the data and the interplay of each component.
- We design a risk assessment framework for IoT to capture potential attack paths across multiple layers.
- We propose an optimization algorithm to reduce the state explosion problem by constructing the hypothesis graph based only on the components relevant to the correctness property specified.
- We investigate the effectiveness of our model by applying it to a realistic example scenario.
- We evaluate the time and space complexity of our framework using the Spin model checker, and the result shows that it only takes seconds and around 100 MB memory to verify hypothesis graph with millions of nodes when there is a violation of the specified correctness property.

We first give a system overview in Section II. Section III presents the formal definition of IoT systems as multilayer state transition graphs. Section IV describes the threat model and problem scope. In Section V, we formalize how to construct the IoT hypothesis graph by combining attacks and the multilayer state transition graph. Section V-C gives a case study of a smart home IoT system and the corresponding hypothesis graph where attacks at different layers are highlighted. Section VII presents our implementation and scalability analysis. Section VIII reviews the related work.

II. System overview

Figure 1 depicts the structure of the framework. First, gather all the components about the target IoT system, including all the physical features (*Env*), user states and behaviors (*Usr*), devices installed (*Dev*), communication events (*Com*), and software applications installed (*App*), and construct the multilayer IoT system transition graph. Then decompose real-world IoT attacks into atomic attacks [7]. From the atomic attacks and the multilayer system transition graph, we build the hypothesis graph and perform vulnerability detection with respect to the specified correctness properties. Finally, if there is a violation of the specified property, an error trace is returned to help us identify the cause. In Section V, we present a state compression algorithm that selects applications and user states relevant to the given correctness property. With the help of this approach, we can collect the relevant components and atomic attacks and directly generate the subgraph of the hypothesis graph for verification.

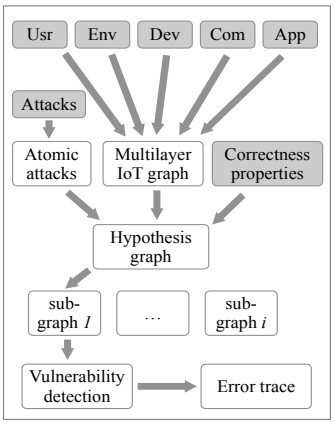

Figure 1. System overview

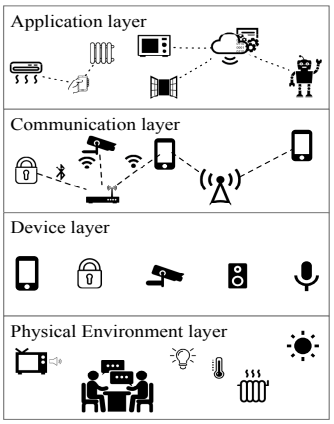

Figure 2. IoT hierarchy

Before constructing the IoT hypothesis graph, we should determine the input of the framework, which are the components with gray box in Figure 1. For a given IoT system, *App* and *Dev* are already known. Then, we can determine *Com* and *Env* based on *App* and *Dev*, since the communication events subscribed or issued by the apps and the mapping between devices and physical features are known. The *Attacks* are derived from the vulnerabilities, which are determined by searching the Common Vulnerabilities and Exposures (CVE) [8] entries for each device and protocol of the system. Once we know the vulnerabilities, we can establish the set of potential attacks on the IoT system. *Correctness properties* are system-specific. Soteria [9] proposed dozens of properties specific to smart home applications and five general properties such as no conflicting control commands or repeated commands in one code branch, etc. However, to the best of our knowledge, currently there is no work that automatically generates correctness properties or selects user states and behaviors.

III. Multilayer State Transition Graph

The heterogeneous nature of the IoT system raises the necessity of a new analysis approach, which not only comprehensively inspects the existing complicated components, but also can be easily extended to expose the vulnerabilities of new devices being plugged, along with the corresponding security and safety impact on the entire IoT system. However, unlike traditional platforms that only involve machines and limited types of user input, an IoT system directly responds to physical environment changes, and can potentially interact with an infinite variety of individual behavior. To capture the extra complexity introduced by additional problem dimensions from IoT, we propose a novel formal framework that abstracts a complicated IoT system into a clear, layered structure. Our

hierarchy-based approach effectively decouples the processing logic of one layer from another so that the vulnerabilities within one layer would not be mixed with others, and makes a complex analysis tractable. Figure 2 illustrates the overview of our IoT hierarchy, which consists of four layers — *physical environment layer, device layer, communication layer*, and *application layer*.

We abstract the internal behavior of each layer as a directed, unweighted state transition graph $L = (V, E)$. In the graph, the node $v \in V$ represents a certain system state of the entire layer. A set of atomic propositions (AP) and their values constitute distinct system states. Each atomic proposition is a boolean variable and describes the smallest unit of the system state that has the characteristic properties of an IoT element. By representing a system state at one layer using a collection of atomic propositions, we make our multilayer state transition graph amenable to model checking algorithms. Sensor measurements of continuous values are discretized into boolean values and represented by atomic propositions as well. For instance, an AP at the device layer describes the value of one temperature sensor. The value of AP is $True$ if the temperature exceeds the threshold $80°$F; otherwise, the value is $False$. We can use the set $v = \{b_{AP_1}, ..., b_{AP_i}, ..., b_{AP_K}\}$ to represent the nodes at a certain layer, where K is the number of APs in a certain layer. The value of b_{AP_i} is either $True$ or $False$. For a layer that has K atomic propositions, there will be 2^K nodes at that layer, representing 2^K system states. Then one edge $e \in E$ describes system state transition. The formal definition of multilayer graph is as follows:

Definition. (Multilayer IoT System Transition Graph) A multilayer IoT system transition graph is a tuple

$$G = (L^{(1)}, \ldots, L^{(4)}, M),$$

where $L^{(i)} = (V^{(i)}, E^{(i)}),\ i = 1, \ldots, 4$, denotes the system state transition sub-graph at physical environment, device, communication or application layer. M is the set of cross-layer edges which indicate the relationships between the adjacent layers and is formally defined as:

$$M = \bigcup_{i \in \{1,2,3\}} (M^{(i,i+1)} \cup M^{(i+1,i)}),$$

where $M^{(i,j)}$ is the set of edges from layer i to layer j.

In the rest of this section, we give detailed definitions of system transition for each layer, along with the node mappings (i.e., cross-layer edges).

Physical environment layer describes the objective fact of physical surroundings and the user states in the IoT system, such as room temperature, humidity, the user being asleep, etc. Here we put the user states in this layer because they also describe the objective fact. The node $v \in V^{(1)}$ represents one specific state of the environment. The edges between nodes denote the system state transition, which may be caused by environmental change or the user's state change.

Suppose v_i and v_j are two nodes at physical environment layer. One atomic proposition AP_l at this layer describes the environmental temperature. If the temperature is larger than threshold θ, the value of b_{AP_l} is $True$, otherwise it is $False$. If the value of AP_l in node v_i is different from the one in node v_j while all of the other atomic propositions are of the same value, then there are two edges of opposite direction between v_i and v_j, implying environmental temperature change.

Let us re-consider the "user and TV" example mentioned in the Introduction section. That the user leaves home without turning off the TV can be represented as an edge between a physical environment-layer node containing $env.user.watch\_TV$ and a device-layer node v_s, where $dev.TV.on$, $dev.presence.false$ and $dev.door.closed$ hold true. Furthermore, the "open the door" voice from TV causes the system state transition from v_s (through communication layer and application layer) to another device-layer node v_t where $dev.TV.on$, $dev.presence.false$ and $dev.door.open$ hold true. In v_t, $dev.presence.false$ and $dev.door.open$ indicate a violation which can be detected by our framework.

Device layer focuses on IoT device status, which is determined by the variable values in the embedded OS of the device. The set of atomic propositions at this layer describes the measurements of environment features and actuator configurations. Some devices can sense the environment, such as the pressure sensor, while the other devices can be configured and operated directly by the user or controlled remotely by software applications, such as an air conditioner or a light bulb. The node $v \in V^{(2)}$ conveys the status of all IoT devices, in terms of atomic propositions and their values. Suppose an IoT device can detect the window state "open or closed", and the value of corresponding atomic proposition AP_k reflects the window state. If two nodes v_i and v_j have distinct $True$ and $False$ values of atomic proposition AP_k, and the other atomic propositions in the two nodes have the same value, then there is an edge to connect these two nodes, indicating a window state change event, such as "opening the window" or "closing the window". If the IoT system functions normally, every edge at application layer corresponds to an edge at device layer, because application commands are delivered to the devices and devices' configuration change are transmitted to the decision maker. The additional edges at device layer indicate some device is compromised, and thus the device status is no longer reported to the decision maker.

There exist cross-layer edges between physical environment layer and device layer, which reflect the route of state transmission. For instance, an edge from physical environment layer to device layer reflects how devices perceive the ground-truth physical state. The nodes $v_i \in V^{(1)}$ and $v_j \in V^{(2)}$ in the two layers have edges if and only if for each atomic proposition $AP_i \in v_i$, all of the associated atomic propositions in v_j have the same value as AP_i. There may be multiple edges connected to one node at physical environment layer, because one environment feature can be measured by multiple devices. For example, humidity can be measured by both thermostat and water leakage sensor. It should be pointed out that environment measurement by IoT devices is not necessarily equal to ground truth at physical environment

layer, as devices could be malfunctioning or compromised.

Communication layer models the events transmitted between devices and the decision makers. Since we consider the most common case in which decision makers reside in the remote cloud which is proprietary and closed-source, we do not model the communication between different decision makers. The events can be categorized into data transmitted from sensors to decision makers, and commands from decision makers to executive devices. The set of the atomic propositions in this layer indicates these events.

The change of information to be transmitted due to sensor measurement is represented as edges in this layer. Suppose v_i is a node where an atomic proposition $humidity \geq 80\%$ holds true, and v_j is a node where the atomic proposition $humidity < 80\%$ holds true. Then the edge between v_i and v_j represents the communication event of information change to be sent by the humidity sensor, due to the humidity decrease.

An upgoing edge from device layer to communication layer indicates that a sensor detects an environmental change and delivers the information to decision makers via transmitting data packets, while a downgoing edge from communication layer to device layer implies a command is delivered to an actuator, causing its configuration change. Due to communication protocol defects or attacks, the communication event may be tampered, thus generating additional edges which lead to some system states that violate the correctness properties.

Application layer formalizes the state of decision makers, which is determined by the set of variable values of software proxies running on the decision making infrastructure. These software proxies act as conduits for physical devices. Hence, the set of atomic propositions in this layer characterizes decision maker's knowledge about the IoT system.

Every node in this layer denotes one particular decision maker state, and an edge represents decision maker state transition due to application rules, or environmental change and actuator configuration change reflected in decision maker's states. Consider room temperature increase causes window open as an example. Suppose the atomic proposition $app.win.closed$ holds true in v_i, while $app.win.open$ hold true in v_j. In particular, $app.temp > 80°\text{F}$ holds true in both v_i and v_j. Then the edge between the two nodes stands for the application rule to open the window when room temperature is higher than $80°\text{F}$.

The edge from communication to application layer signifies that the event packets sent by the sensor are faithfully delivered to the decision maker, triggering the update of variable value in decision maker. Similarly, an edge from application layer to communication layer indicates that the decision maker' state is updated due to the application rules, and it also generates command packets to be sent to the actuator(s).

Only verifying that a system does not satisfy the property is not sufficient; we should also visit back to identify the root causes of attacks. In our framework, the interconnection among the layers is explicitly captured by their node mappings, which helps trace the influences from one layer to another and finally identify the propagation path a venerability.

Table I
TYPICAL IOT ATTACKS HAPPENING AT DIFFERENT LAYERS

Attack	Env	Dev	Com	App
Mirai [3]		✓	✓	
IoTMON [10]	✓			✓
Sniffing attack [6] [11]			✓	
Rocking drones [12]	✓	✓		
Soundcomber [13]		✓	✓	
Vampire attack [14]		✓	✓	

IV. THREAT MODEL

In this paper, we consider IoT system vulnerabilities (integrity violations) caused by flawed or malicious apps, user's behaviors, attacks, or their interactions via common channels such as physical environment features or shared devices. Due to the distributed and heterogeneous nature of the IoT systems, such violations are difficult to predict. To analyze the attacks' impact on system security, we first need to integrate them into the system transition graph. While some real-world attacks to IoT systems happen at only one layer, many others involve multiple steps at different layers. We follow [7] and name every single step an atomic attack. Table I surveys typical attacks and layers at which they operate.

Furthermore, we consider both passive attacks and active attacks that happen at all of the four layers of the IoT system. It is assumed that the attacker is aware of the commercial IoT system architecture. Besides, he knows the communication between the gateway and the cloud; he also knows the protocols used for inter-device communication as they are industry standards. The attacker's arsenal is all the vulnerabilities listed on Common Vulnerabilities and Exposures (CVE) of all the devices installed and protocols used. We assume that the remote cloud is trustworthy and do not attempt to model attacks on the cloud.

V. HYPOTHESIS GRAPH

Due to the interactive nature of IoT components, attacks may trigger unexpected security issues. Thus, it is necessary to model the attacker's behavior and integrate it into the IoT system model to construct a novel, more realistic state transition graph amenable to existing formal verification tools. We name our multilayer IoT state transition graph with attacker behaviors as *hypothesis graph*. The formal definition of hypothesis graph is given below.

Definition. (IoT Hypothesis Graph) An IoT hypothesis graph is a multilayer graph $G = (L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)}, M)$, where $L^{(l)} = (V^{(l)}, E^{(l)}),\ l \in \{1, 2, 3, 4\}$ is state transition graph at layer l and M is the node mapping. Each node $v \in V^{(1)} \cup \cdots \cup V^{(4)}$ denotes the system and the *attacker's state*. Each edge $e \in E^{(1)} \cup \cdots \cup E^{(4)} \cup M$ denotes environmental change, user's behavior, information flow, or *attacker's behavior*.

Compared with the multilayer IoT system transition graph, the hypothesis graph contains extra atomic propositions for attacker's states, which can appear at all of the layers except

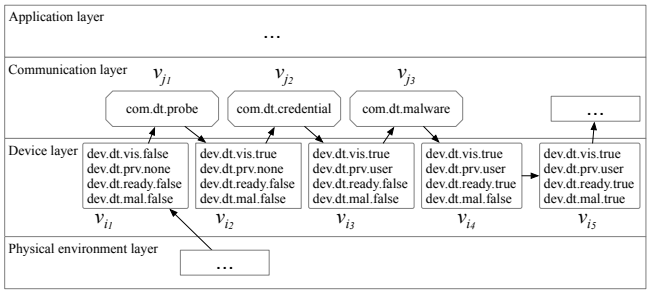

Figure 3. The illustration of Mirai cross-layer attack

the environment layer, and additional edges for attacker's behaviors, which are across the device and communication layer, across communication and application layer, or within the device layer or physical environment layer.

A. Modeling Attacker Behavior

As is shown in section IV, a real, complete attack may involve multiple atomic attacks. So from now on, when talking about attacks, we mean atomic attacks. To model passive attacks (which do not change system configuration), we introduce atomic propositions associated with devices' or events' **visibility** to the attacker. To model active attacks (which change the system configuration), we introduce atomic propositions associated with the services running on a device and attacker's **privilege** on a device. We assume an attacker may have one of the three privileges on a device: $none$, $user$, and $root$. To model attacks via the network, we add atomic propositions to represent malware or other packets generated by the attacker such as username-password pair.

Here we use Mirai attack as an example and show how to decompose it into atomic attacks and represent each atomic attack. In Mirai attack, the device's infection mechanism can be decomposed into the following four steps — scanning the potential victim, brute-force login, malware dispatch, and malware execution. The first three steps happen at the communication layer, while the last one is at the device layer. Assuming the victim device is d_t, Figure 3 illustrates the cross-layer attack. Node $v_{i_1} \sim v_{i_5}$ are device-layer nodes representing system and attacker's states before or after the atomic attack. $v_{j_1} \sim v_{j_3}$ are communication-layer nodes representing probing packets, credential, or malware image, respectively. For clarity, we only list APs that are relevant to the Mirai attack. The values of all the other APs in $v_{i_1} \sim v_{i_5}$ are the same.

Device scanning: In this step, the attacker sends TCP SYN probes to pseudorandom IPv4 addresses on Telnet TCP ports. If the device d_t responds, then the attacker knows the existence of d_t, i.e., the device becomes visible to the attacker. This is reflected as the AP value change from $dev.d_t.vis.false$ to $dev.d_t.vis.true$. The atomic attack is represented as two added edges (v_{i_1}, v_{j_1}) and (v_{j_1}, v_{i_2}).

Brute-force login: The attacker attempts to log in to the device by trying 10 different credentials. A successful login give the attacker $user$ privilege, which is reflected as AP value change in v_{i_2} and v_{i_3}. The attack behavior is also represented as two added edges (v_{i_2}, v_{j_2}) and (v_{j_2}, v_{i_3}).

Malware dispatch: After logging in to the victim device, the attacker checks the system environment, including OS version and CPU architecture, etc., and then download the malware binary image. Similar to the previous two steps, we use two APs to represent the system state before and after the attack. More specifically, $dev.d_t.ready.false$ holds true in v_{i_3} (meaning the device is not ready for the launch of the malware image), while $dev.d_t.ready.true$ holds true in v_{i_4}. Edge (v_{i_3}, v_{j_3}) and (v_{j_3}, v_{i_4}) model this atomic attack.

Malware execution: This step is the loading and execution of malware binary image. The AP $dev.d_t.mal.true$ in v_{i_5} indicates the malware process is running. The atomic attack is represented as the edge (v_{i_4}, v_{i_5}). Once executed, the malware performs a sequence of sabotage such as obfuscating its process name, killing other processes, or privilege escalation, etc. All these malicious behaviors are represented as additional edges that follow node v_{i_5}.

Many real-world IoT attacks can be decomposed as atomic attacks mentioned above. Our added atomic propositions make sure the correct sequence of atomic attacks which should be followed by the attacker. For example, the attacker should first sniff the existence of a device; only then can he launch the remote-to-user attack. To formally define an atomic attack, we need to identify the system and attacker states before and after the attack. Then the attack behavior is represented as the added edge between these two nodes.

B. Constructing Hypothesis Graph

As is shown in Section V-A, for some attack, we need to introduce new atomic propositions (e.g., $dev.dt.vis.false$ and $dev.dt.prv.none$, etc.) to represent the attack. In this subsection, we define a basic operation named *state expansion* to show how to accommodate the newly inserted atomic propositions. After that, we can represent all the attack behavior as added edges and construct the final hypothesis graph.

State expansion. Suppose we are trying to insert an atomic proposition ap to a certain layer and the original graph of this layer has $|V|$ nodes and $|E|$ edges. After state expansion, the new graph for this layer has $\left|V'\right|$ nodes and $\left|E'\right|$ edges. If ap is independent of all the existing atomic propositions, then we have $\left|V'\right| = 2 \times |V|$ and $\left|E'\right| = 2 \times |E|$. Formally, when we try to insert an atomic proposition ap to layer l, first duplicate the original graph of layer l (The cross-layer edges are also duplicated.), then make all of the nodes of one copy have ap being *True*, while the other copy have ap being *False*.

After state expansion for all of the attacks which require additional atomic propositions, we can safely add edges to represent attack behaviors. The resulting graph is an IoT hypothesis graph whose nodes depicts system states including the attacker's state at certain layer and whose edges represent state transition due to environmental change, user's behavior, information flow, or attacker's behavior.

Algorithm 1: Dynamic Selection Algorithm

Input: $S_{App} = \{App^{(1)} \ldots, App^{(n)}\}$: the set of all the apps installed and the virtual apps representing user states and behaviors.
p: the correctness property specified.

Output: $S \subseteq S_{App}$: the set of all the apps that should be considered together for the specified correctness property.

Algorithm `dynamic_selection`(S_{App}, p)
  Construct the virtual app $App^{(0)}$ by determining the set $E_{in}, E_{out}, A_{in},$ and A_{out} from the given correctness property p.
  $S = \{App^{(0)}\}$
  `/* Iteratively add related apps to` S. `*/`
  $old\_size = |S|$
  do
    $T = S_{App} \backslash S$
    for $x \in T$ **do**
      for $y \in S$ **do**
        if `is_related`(x, y) **then**
          $S = S \cup \{x\}$
        end
      end
    end
    $new\_size = |S|$
  while $new\_size \neq old\_size$
  $S = S \backslash \{App^{(0)}\}$
  return S

Procedure `is_related`(x, y)
  `/* Determine if app` x `and` y `are related.` `*/`
  return $x[E_{in}] \cap y[E_{out}] \neq \varnothing$ **or** $x[E_{out}] \cap y[E_{in}] \neq \varnothing$ **or** $x[E_{out}] \cap y[E_{out}] \neq \varnothing$ **or** $x[A_{in}] \cap y[A_{out}] \neq \varnothing$ **or** $x[A_{out}] \cap y[A_{in}] \neq \varnothing$**or** $x[A_{out}] \cap y[A_{out}] \neq \varnothing$

C. Vulnerability Detection

Our framework is built on model checking, which takes system graph and correctness properties as input, and outputs a counterexample if the system does not satisfy a certain correctness property.

Specifying correctness property. Correctness properties for a system can be classified as safety property (that something "bad" will never happen) and liveness property (that something "good" will eventually happen). Here we express correctness properties using Linear Temporal Logic (LTL) syntax [15].

Below are some examples of LTL formula and their meanings (We follow the convention and omit the leading **A** in each LTL formula):

- $\mathbf{G}(dev.user.state.u_0 \rightarrow dev.alexa.off)$ means the voice assistant should be off if the user is not at home.

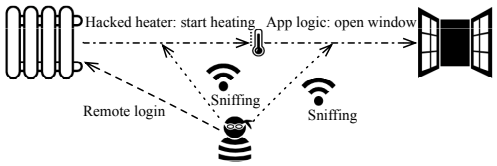

Figure 4. Example of smart home attack

- $\mathbf{G}\neg(dev.user.state.u_2 \wedge dev.door.open)$ means it should never happen when the user is sleeping and the door is open.
- $\mathbf{G}(camera.prv.none)$ means the attacker should never gain access to the surveillance camera.

Model checking. Though there are many model checking algorithms, their inputs all originate from Kripke structure [15]. Our multilayer hypothesis graph conforms to the definition of Kripke structure, and thus is applicable to existing model checkers. Specifically, the model checking algorithms based on automata theory first transform the Kripke structure into a finite automaton, then transform the negation of the correctness properties into a Büchi automaton. After that, they compute the intersection of the above two automata and return a counterexample if the intersection is not empty.

D. State Space Compression

A major challenge of model checking is the state explosion problem. Though introducing user and attack can make the system model more realistic, the number of nodes of the model gets 2^k (k is the number of newly introduced atomic propositions to represent user and attacker's states) times bigger, thus worsening the state explosion problem. Therefore, we propose a dynamic selection algorithm that selects relevant applications and user states, given the correctness property. Because the algorithm is executed before constructing the hypothesis graph, it can be used regardless of the model checking algorithm chosen.

Our algorithm is based on the observation that every correctness property or IoT application involves environment features and/or actuator configurations. Moreover, each user state and associated behavior can also be seen as a *virtual* application. Hence, formally any given application i can be represented as

$$App^{(i)} = (E_{in}^{(i)}, A_{in}^{(i)}, E_{out}^{(i)}, A_{out}^{(i)}),$$

where $E_{in}^{(i)}$ is the set of input environment features (including user states), $A_{in}^{(i)}$ is the set of input actuator configuration, and $E_{out}^{(i)}$ and $A_{out}^{(i)}$ are the output counterparts. For a virtual app of user state and behaviors, E_{in} is the set of the current user state, $A_{in} = \varnothing$, E_{out} is the set of all of the possible next state from current state, and A_{out} is the set of all the possible next actuator configuration due to user's behavior in current user state. Then we can determine whether any two given apps are related or not using the following rule: If one's output environment feature/actuator configuration is used by the other as input, or if the two apps have common output environment feature/actuator configuration, then they are related.

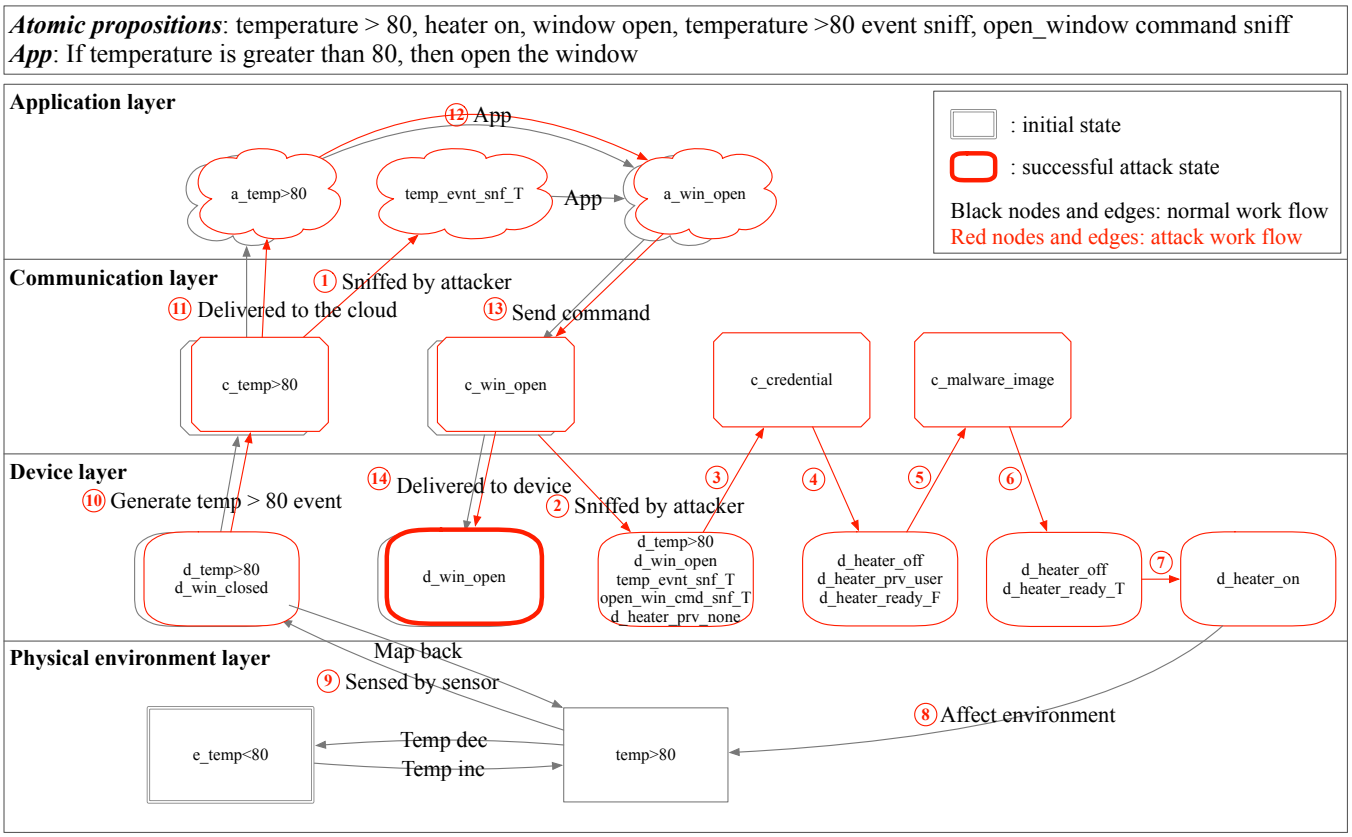

Figure 5. The hypothesis graph with attack trace

The algorithm is shown in Algorithm 1. Given the pool of applications and user data, along with the specified correctness property, the algorithm starts from environment features and actuator configurations used by the property as seeds, then iteratively marks applications (including virtual apps) until the set of marked apps does not change. The subroutine `is_related`(x, y) determines whether app x and y are related. In Line 3, it puts $App^{(0)}$ as the seed. After we put all the related apps into S, we can remove $App^{(0)}$ (Line 16).

VI. Case Study

In this section, we present a proof-of-concept attack inspired by [3, 10, 6] and the corresponding IoT hypothesis graph. The attacks cross device, communication, and application layer. The attacker's goal is to break into a smart house. The smart house is equipped with a heater and an automatic window. Among the software applications installed, there is one particular app — If the temperature is greater than the threshold, then turn on the heater. The IoT setting and the attacker are illustrated in Figure 4. The safety properties is expressed in LTL syntax as

$$\boldsymbol{G}(dev.window.open \rightarrow \neg dev.user.state.u_0).$$

Figure 5 shows the corresponding hypothesis graph for the scenario. For clarity, we only label each node with atomic propositions whose value get changed from the preceding node. The final red node denotes the violation of the property, i.e., the attacker's goal is achieved, and the label for each edge shows the cause of the state transition. Notice that there could be multiple paths connecting the same starting and ending node and here we are only showing one path for illustration.

The atomic proposition in the bottom left initial state tells us that the room temperature is less than the threshold. The increase of room temperature is sensed by the temperature sensor, and the sensor generates a wireless event (represented by the communication layer node with the atomic proposition *c_temp>80*). This wireless event is sniffed by the attacker, whose sniffing behavior is denoted as edge ①. The decision maker receives the event and updates the variable values in the software proxy. The App logic controls the window to open by sending the window open command (denoted by the node labeled with *c_win_open*) to the window. This control signal is also sniffed by the attacker (labeled by ②), and thus cause the atomic proposition *temp_evnt_snf* and *open_win_cmd_snf* to hold true. Edge ③∼⑦ represent the brute-force login, malware dispatch, and malware execution, as explained in Section V-A. Edge ⑧∼⑭ denote the attacker's exploitation of the system's vulnerability to force the window open.

The model checking algorithm first determines that the node with a thick red border is a state which violates the specified correctness property. Then it traces back and marks all the preceding nodes until it reaches the initial node. Since from

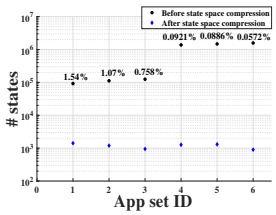

Figure 6. Number of states before and after compression and the compression ratio.

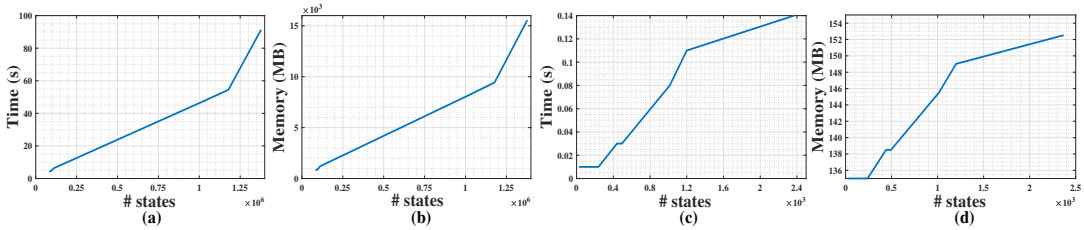

Figure 7. Impact of hypothesis graph size (measured by number of states) on verification time and memory usage. 7 (a) and 7 (b) show the scenarios where the models pass the verifier; 7 (c) and 7 (d) show the scenarios where there are violations to the property.

the initial system state we can finally reach the violating state, the hypothesis graph does not satisfy the specified correctness property, and the error trace is the red path in Figure 5.

VII. Evaluation

A. Implementation

We implement ForeSee based on the Spin model checker [16]. Spin takes a system modeling language called Promela (Process Meta Language) [16] as its input language, and accepts correctness properties specified as Linear Temporal Logic (LTL) formulas. We use IoTSan to convert SmartApps to Promela language, then modify the Promela code by inserting variables for physical environment, device, and communication layer, as well as atomic attacks. After that, we perform verification by running the compiled program. If the system does not satisfy the given correctness property, the verifier will return an execution trace that caused the violation.

B. Performance Analysis

For a given LTL property, we test the scalability of our framework under two different settings: 1) when our system passes the verifier; 2) when there is a violation of the property. The time and space complexity of hypothesis graphs that pass the verifier are shown in Figure 7(a) and 7(b), while the ones of the hypothesis graph that fail to pass are shown in Figure 7(c) and 7(d). The x-axis variable "# states" denotes the number of unique states of the hypothesis graph traversed by Spin model checker. This is used as a measure of the size of the hypothesis graph. The y-axis variable "Memory (MB)" in Figure 7 denotes the sum of memory used to store all these states, hash table, depth-first search stack, and other overhead. Due to the server's memory limit, the maximum number of states we can run is around 1.4×10^6. Since Spin will immediately return after detecting a violation (i.e., an acceptance cycle), the verification process takes much less time and space if there is a violation. The scalability of our framework when there exists a violation is shown in Figure 7. From the figures, we can see that the time and space cost scale linearly with the graph size, and it takes much less time and memory when there is a violation of the property.

VIII. Related Work

The research works on IoT security and privacy can be categorized by the components they focus on.

Device Layer. Costin *et al.* [17] conducted a static analysis of 32 thousand embedded firmware images and discovered 38 previously unknown vulnerabilities of embedded devices. Son *et al.* [12] presented real-world attacks to drones by employing the resonant frequencies of Micro-Electro-Mechanical Systems gyroscopes. Ronen *et al.* [18] described an attack on Philips Hue smart lamps by exploiting a major bug in their implementation of the ZigBee protocol.

Communication Layer. Gu *et al.* [19] proposed a defense framework against device spoofing attacks by fingerprinting and authenticating IoT devices using features generated from Bluetooth low energy protocol stack. Jia *et al.* [20] presented a graph-based mechanism to detect vulnerabilities in IoT communications by rating and sorting the correlated subgraphs extracted from the directed graph generated from the traffic data. Li *et al.* [21] investigated the side-channel information leakage of video surveillance cameras through streaming traffic data analysis.

Application Layer. Ding *et al.* [10] presented a framework that discovers potential physical interactions across applications using natural language processing (NLP) techniques and evaluated the risk score of each inter-app interaction chain. Mohsin *et al.* [22] proposed a formal framework for IoT security analysis based on satisfiability modulo theories (SMT). [9, 23] took advantage of model checking to analyze application-level vulnerabilities in IoT systems. Mohsin *et al.* [24] presented a probabilistic model checking based framework to analyze the risks quantitatively.

IX. Conclusion

In this paper, we design and prototype ForeSee , a cross-layer vulnerability analysis framework for IoT systems. We propose a formal approach to construct the IoT hypothesis graph which includes all of the core components of IoT systems, including user states and behavior that are largely ignored in existing works, and the potential attacks. The framework detects vulnerabilities and threats caused by any interaction between those components. Besides, a dynamic state space compression algorithm is presented. Our evaluation suggests that the framework scales well on hypothesis graphs consisting of millions of nodes. And the compression algorithm is able to reduce the number of states by three orders of magnitude.

X. Acknowledgement

This research was sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] I. Lunden, "IFTTT raises $24m led by salesforce to expand its platform to 'connect everything'." https://techcrunch.com/2018/04/26/ifttt-raises-24m-led-by-salesforce-to-expand-its-platform-to-connect-everything/.

[2] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon, "Unlocking the potential of the internet of things," *McKinsey Global Institute*, 2015.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium*, pp. 1093–1110, 2017.

[4] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *26th USENIX Security Symposium*, pp. 361–378, 2017.

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

[6] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pp. 1074–1088, 2018.

[7] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings 2002 IEEE Symposium on Security and Privacy*, pp. 273–284, 2002.

[8] "Common vulnerabilities and exposures." https://cve.mitre.org/.

[9] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated IoT safety and security analysis," in *2018 USENIX Annual Technical Conference*, pp. 147–158, 2018.

[10] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pp. 832–846, 2018.

[11] N. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic," *arXiv preprint arXiv:1705.06805*, 2017.

[12] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *24th USENIX Security Symposium*, pp. 881–896, 2015.

[13] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones.," in *NDSS*, vol. 11, pp. 17–33, 2011.

[14] E. Y. Vasserman and N. Hopper, "Vampire attacks: Draining life from wireless ad hoc sensor networks," *IEEE Transactions on Mobile Computing*, vol. 12, pp. 318–332, Feb 2013.

[15] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.

[16] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*, vol. 1003. Addison-Wesley Reading, 2004.

[17] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *23rd USENIX Security Symposium*, pp. 95–110, 2014.

[18] E. Ronen, A. Shamir, A. Weingarten, and C. O'Flynn, "Iot goes nuclear: Creating a zigbee chain reaction," in *2017 IEEE Symposium on Security and Privacy*, pp. 195–212, 2017.

[19] T. Gu and P. Mohapatra, "BF-IoT: Securing the iot networks via fingerprinting-based device authentication," in *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 254–262, 2018.

[20] Y. Jia, Y. Xiao, J. Yu, X. Cheng, Z. Liang, and Z. Wan, "A novel graph-based mechanism for identifying traffic vulnerabilities in smart home iot," in *IEEE INFOCOM*, pp. 1493–1501, 2018.

[21] H. Li, Y. He, L. Sun, X. Cheng, and J. Yu, "Side-channel information leakage of encrypted video stream in video surveillance systems," in *IEEE INFOCOM*, pp. 1–9, 2016.

[22] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman, "IoTSAT: A formal framework for security analysis of the internet of things (IoT)," in *2016 IEEE Conference on Communications and Network Security (CNS)*, pp. 180–188, 2016.

[23] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. M. Colbert, and P. McDaniel, "IotSan: Fortifying the safety of iot systems," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, pp. 191–203, 2018.

[24] M. Mohsin, M. U. Sardar, O. Hasan, and Z. Anwar, "IoTRiskAnalyzer: A probabilistic model checking based framework for formal risk analytics of the internet of things," *IEEE Access*, vol. 5, pp. 5494–5505, 2017.