A Model Checking-Based Security Analysis Framework for IoT Systems

Zheng Fang[a], Hao Fu[a], Tianbo Gu[b], Zhiyun Qian[c], Trent Jaeger[e], Pengfei Hu[d], Prasant Mohapatra[a]

[a] *University of California, Davis*
[b] *Microsoft*
[c] *University of California, Riverside*
[d] *Shandong University*
[e] *Pennsylvania State University*

Abstract

IoT systems are revolutionizing our life by providing ubiquitous computing, inter-connectivity, and automated control. However, the increasing system complexity poses huge challenges for security as IoT devices are distributed, highly heterogeneous, and can directly interact with the physical environment. In IoT systems, bugs in device firmware, defects in network protocols, and design flaws in automation rules can lead to system breach or failure. The challenge gets even more escalated as the possible attacks may be chained together in a long sequence across multiple layers, rendering the existing vulnerability analysis frameworks inapplicable. In this paper, we present ForeSee, a model checking-based framework to comprehensively evaluate IoT system security. It builds a multi-layer *IoT hypothesis graph* by simultaneously modeling all of the essential components in IoT systems, including the physical environment, devices, communication protocols, and applications. The model checker can then analyze the generated hypothesis graph to validate system security properties or generate attack paths if there are any violations. An optimization algorithm is further introduced to reduce the computational complexity of our analysis. Our framework verifies hypothesis graphs with millions of nodes in less than 100 seconds. The illustrative case studies show that our framework can detect more potential threats than the existing approaches.

Keywords: IoT, security and privacy, model checking, hypothesis graph

1. Introduction

Nowadays, Internet of Things (IoT) systems are deployed in a wide range of applications: smart home, industrial manufacturing, healthcare, transportation, and in many other sectors [1]. There are already 20 billion IoT devices connected to the internet, and the number is expected to rise to 75 billion by the year 2025 [2]. Rapid growth of IoT market has evolved IoT technologies. Various communication protocols, applications, and platforms are designed for diverse application scenarios. Popular IoT platforms, such as Samsung SmartThings [3], Google Nest [4], and IFTTT [5], etc., attract more and more developers to develop numerous applications to automate our life. For example, there are more than 5,000 active developers and 75 million Applets since the launch of IFTTT platform [6].

With the wide deployment of IoT systems comes an increasing number of IoT attacks. Existing IoT security research focuses on subsets of all of the core components in IoT [7, 8, 9, 10]. However, the heterogeneity and interaction between different IoT components require a cross-layer framework which considers IoT system in a holistic view. For example, researchers at Pen Test Parterners have found that attackers can exploit communication protocol vulnerabilities to change the physical state of an IoT system, such as unlocking the doorlock [11] and increasing the temperature of a hair straightener [12]. Making the matters worse, the physical state change can be sensed by sensors, further triggering other actuator behavior. Even though recent IoT security analysis frameworks [13, 14] claim to be cross-layer, they still focus on a subset of IoT system components, such as communication protocol stack, failing to consider other essential components such as users' behavior, physical environment, and IoT applications. However, an increasing number of IoT attacks and threats [15, 16, 17] shows that cyber attack can lead to physical breach, and vice versa.

To solve these challenges, we present ForeSee, a cross-layer security analysis framework, to treat IoT system security from a holistic perspective. We first

decouple and model an IoT system as a multi-layer graph, including *physical environment layer*, *device layer*, *communication layer* and *application layer*. The multi-layer graph models both intra-layer and inter-layer interaction between different components. Furthermore, FoReSee decomposes real-world IoT attacks into individual exploits and integrates them into the multi-layer graph, generating attack traces to show how they interact with system components.

The benefits of our approach are threefold. First, by considering all of the core components simultaneously, we can discover more vulnerabilities than existing frameworks do. For example, suppose an incompetent user is surfing the internet at home, and the indoor camera is running a vulnerable network service. If the user clicks the phishing site created by the attacker, the camera will be exploited. The attacker can use the compromised camera to further spoof a "intruder detected" event to trigger the alarm or other unwanted device behavior. Existing works fail to model user state or behavior and thus will not be able to detect such an attack path.

Second, we identify and model various device interactions. For instance, if an air conditioner is plugged into a smart outlet and the outlet has a denial-of-service (DoS) vulnerability, then the attacker can disable the air conditioner by launching a DoS attack on the outlet. As a result, the AC will also be off, failing to cool the room. This may even trigger other potential actions such as opening the window, etc. Frameworks focusing solely on software applications cannot discover such vulnerabilities because they involve electrical dependence between devices. Lastly, we can examine how seemingly unimportant vulnerabilities escalate and cause disastrous results due to the interactive nature of the IoT devices. This helps us better evaluate the vulnerabilities' impact on system security and prioritize the protection against them.

We create a multi-layer graph by defining system states at different layers using boolean variables and formulating state transitions within each layer and between adjacent layers. After generating the multi-layer IoT system graph, we explore all the existing and potential attacks and incorporate them into the graph to form a final *hypothesis graph*. Then we apply model checking

technique to detect various vulnerabilities and attacks. To alleviate the size explosion problem of the hypothesis graph, we design a state compression algorithm to intelligently generate independent sub-graphs without compromising the vulnerability detection ability.

In summary, we make the following contributions:

- We formally represent IoT systems as multi-layer graphs to characterize data flow and the interaction of different components.
- 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 using a case study which is based on real-world IoT attacks.
- We evaluate the time and space complexity of our framework using the SPIN model checker [18], 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.

2. Background

2.1. Threats to IoT Systems

IoT systems connect physical world to the cyber space. Today's typical IoT systems have complex infrastructure including router(s), gateways (sometimes called hubs, basestation, etc.), end devices, and a cloud backend. Usually there is also a companion mobile app for remote control. The end devices can be categorized as sensors and actuators to perceive and modify physical states of the system. However, numerous vulnerabilities have been found on IoT devices [19, 16, 20, 17, 21, 22] and mobile apps [23, 24]. Another important feature of

Table 1: Typical IoT attacks happening at different layers.

Attack	Env	Dev	Com	App
Mirai [7]		✓	✓	
IoTMON [27]	✓			✓
Sniffing attack [10] [28]			✓	
Rocking drones [19]	✓	✓		
Soundcomber [29]		✓	✓	
Vampire attack [30]		✓	✓	

IoT is that users can install IoT applications for automatic control. These IoT apps usually run in the cloud and use the trigger-action programming paradigm. The trigger is some IoT event represented as device state change, for example, the thermostat senses an increase of environment temperature, or the door lock is unlocked. The action represents some device behavior, such as turning on the light or sounding the alarm. Even though IoT apps' logic seems straightforward, researchers have identified dozens of malicious IoT apps which may cause system breach or other physical damage.

In addition, there are some human users interacting with the IoT system. For example, the user's existence in a smart home will be sensed by a motion sensor, and the user can take actions such as turning on the TV or opening the window. However, existing works [25, 26] do not model the user's real physical state and only use sensors' input as true user states. This may result in false negatives because the attacker can spoof sensor events [16]. Therefore, in order to detect threats to users' safety in IoT systems, we need to distinguish between users' true, physical state and sensors' reported user state, and integrate both into the model. Table 1 surveys typical attacks and layers at which they operate.

2.2. Model Checking

Model checking is a formal verification technique used to automatically verify whether a system satisfies the specified property by exhaustively searching all of the reachable states of the system model [31]. Our project is based on explicit-

state model checking, which represents a finite-state system as a state transition graph, and uses temporal logic to specify properties to be verified. We choose explicit-state model checking because: (i) the state transition graph can be easily extended to incorporate environment and user states which are essential for IoT security analysis, (ii) temporal logic formulas are powerful for expressing correctness properties, and (iii) the error trace returned by a model checker can help us quickly identify the root cause of the problem.

The state explosion problem has been the greatest challenge to model checking, and due to the large number of devices and other attributes such as the physical environment features, this issue is only getting more serious for IoT systems. To mitigate this challenge, we design an algorithm to reduce the number of states in the system model without affecting the capability of the model checker for the given correctness property. Moreover, granularity is also important when modeling the system. Some IoT features such as room temperature is a continuous number, but to make the system state finite, we need to discretize it or even make it boolean. There is a tradeoff between the number of potential vulnerabilities we can detect and the time and memory usage.

Due to its popularity and high efficiency, we choose SPIN model checker [18] to verify our hypothesis graph. SPIN accepts PROMELA [32] as system description language and linear temporal logic (LTL) formulas as correctness properties to be verified.

3. 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 [33] and name every single step an atomic attack.

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, the attacker knows the communication between the gateway and the cloud; attacker 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) [34] 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.

4. System overview

Figure 1 depicts the structure of the framework. First of all, we gather all of the components of 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*). Then we construct the multi-layer IoT system transition graph. Thereafter, we decompose real-world IoT attacks into atomic attacks [33]. From the atomic attacks and the multi-layer 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 6, 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.

Before constructing the IoT hypothesis graph, we should determine the input of the framework shown as gray boxes in Figure 1. For a given IoT system, *App* and *Dev* are already known. Then, we can determine *Com* and *Env* based

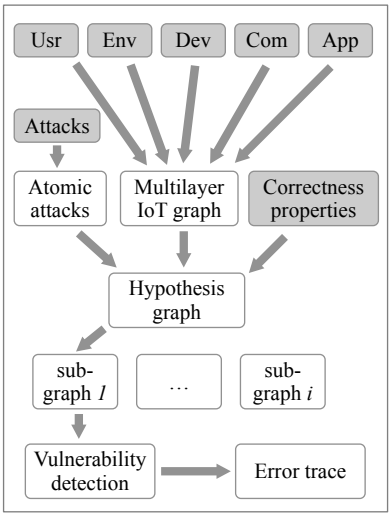

Figure 1: System overview.

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) [34] 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 [35] 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 comprehensively deals with user states and behaviors.

5. Multi-Layer State Transition Graph

5.1. Multi-Layer Graph Construction

The heterogeneous and dynamic nature of IoT systems brings huge challenges for system security analysis. First of all, IoT systems directly interact

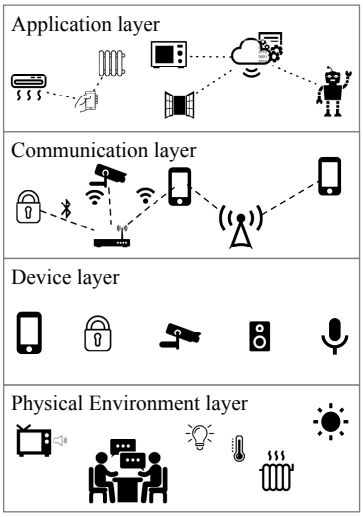

Figure 2: IoT hierarchy.

with the physical environment, and can potentially interact with an infinite number of user states and behavior. Second, devices may be added to or removed from the IoT system frequently. Moreover, the framework should consider both system security and user safety as they are an essential part for most of IoT systems. To deal with the above challenges, we propose a novel formal framework that abstracts a complicated IoT system into a clear, layered structure. Our approach effectively decouples the processing logic of one layer from another so that the vulnerabilities within one layer would not be mixed with others. Moreover, multi-layer graph also enables us to detect violations which involve multiple layers, such as inconsistency between physical and device layer. Figure 2 gives an overview of the 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) [31] 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 multi-layer 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 multi-layer graph is as follows:

Definition. (Multi-Layer IoT System Transition Graph) A multi-layer 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 facts about 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 will be an edge from v_i to v_j and an edge from v_j to v_i, 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°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°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.

5.2. IoT App Description Analysis

Since IoT applications decide the functionality of an IoT system and they are dependent on the users' configuration, we need to design an approach to automatically extract the app logic based on app description or the app source code. Our method extracts apps' semantic information from their descriptions using NLP techniques. A typical IoT app description is in trigger-action format where the trigger is some IoT event and the action means some device behavior.

First of all, we use an NLP parser to construct the parse tree and split the sentence into the conditional clause and the main clause by doing a breadth-first search (BFS) on the parse tree to find the tree node with label `SBAR`, which is the root of the subtree for the conditional clause. Then the conditional clause is obtained by concatenating the leaf nodes of this subtree. The main clause is constructed by removing the conditional clause from the original description. After that, we extract the noun and verb phrases from each clause using regular expression chunker and match the noun and verb phrases with device name and device actions, respectively. The matching is based on Word2Vec embedding [36]. Because the embedding is only for individual words, we split every phrase into words and choose the highest word pair similarity as the match result.

As an example, the parse tree of an IoT app description is shown in Figure 3. The conditional and main clauses after splitting are "motion detected" and "turn on light for 10 minutes." The regular expression patterns for chunking is shown in Listing 1. The final extracted app logic can be represented as a Python dictionary shown in Listing 2.

```
NP: {<DT>?<JJ>*<NN.*>+}
VP: {<VB.*><IN|RP>?}
```

Listing 1: Regular expression patterns for chunking.

```
{'conditional': (['motion sensor'], ['motion']), 'main': (['bulb'],
     ['on'])}
```

Listing 2: Internal representation of an IoT app logic.

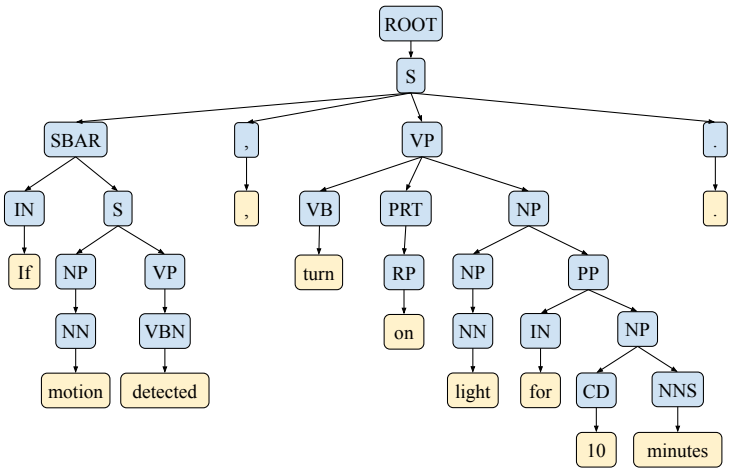

Figure 3: The constituency parse tree of the app description "If motion detected, turn on light for 10 minutes."

Based on the extracted IoT app logic and IoT system configuration information, we can get the state-transition graph for the application layer. Since the mapping of atomic propositions from one layer to another are straightforward, we can decide the cross-layer edges and the set of atomic propositions to be considered for the other three layers. Last, we need to decide the edges within the device and physical layer. The physical environment change are consistent in all of the IoT systems (For example, for all the IoT systems which consider environment temperature, there is an edge labeled with *temperature increase* from the node which contains *env.temperature.low* to the one which contains *env.temperature.high*) and thus can be pre-defined. The app logic will also be used by the dynamic selection algorithm (explained in Section 6.4).

6. 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 multi-layer 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 multi-layer 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 multi-layer IoT system transition graph, the hypothesis graph contains extra atomic propositions for attacker's states, which can appear at all of the layers except 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.

6.1. Modeling Attacker Behavior

A real, complete attack may involve multiple 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

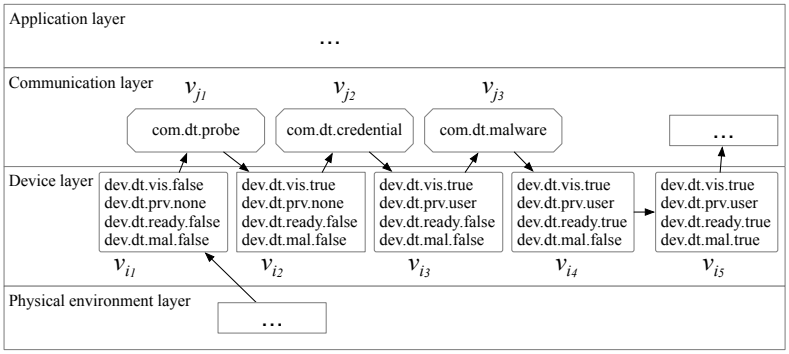

Figure 4: The illustration of Mirai cross-layer attack.

the last one is at the device layer. Assuming the victim device is d_t, Figure 4 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.

6.2. Constructing Hypothesis Graph

As is shown in Section 6.1, 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. Then we represent the attack behavior as 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

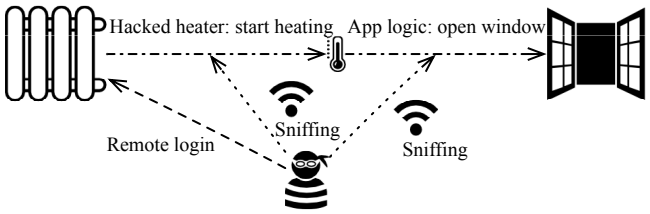

Figure 5: Example of smart home attack.

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.

6.3. Vulnerability Detection

Our framework is built on top of 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), which are expressed as Linear Temporal Logic (LTL) formulas [31].

Model checking. Though there are many model checking algorithms, their inputs all originate from Kripke structure [31]. Our multi-layer hypothesis graph conforms to the definition of Kripke structure, and thus is applicable to existing model checkers.

6.4. 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.

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 by checking whether one's output environment feature or actuator behavior is the other's input. If one of the set intersection is non-empty (meaning components interact with each other), there is a dependency. 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).

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.

```
1  Algorithm dynamic_selection(S_App, p)
2      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.
3      S = {App^(0)}
       /* Iteratively add related apps to S.                    */
4      old_size = |S|
5      do
6          T = S_App \ S
7          for x ∈ T do
8              for y ∈ S do
9                  if is_related(x, y) then
10                     S = S ∪ {x}
11                 end
12             end
13         end
14         new_size = |S|
15     while new_size ≠ old_size
16     S = S \ {App^(0)}
17     return S

1  Procedure is_related(x, y)
       /* Determine if app x and y are related.                 */
2      return x[E_in] ∩ y[E_out] ≠ ∅ or x[E_out] ∩ y[E_in] ≠ ∅ or
       x[E_out] ∩ y[E_out] ≠ ∅ or x[A_in] ∩ y[A_out] ≠ ∅ or
       x[A_out] ∩ y[A_in] ≠ ∅or x[A_out] ∩ y[A_out] ≠ ∅
```

7. Case Study

We illustrate our framework's wide applicability by designing hypothesis graphs for two IoT systems: smart home and smart healthcare.

7.1. Smart Home

In this subsection, we present a proof-of-concept attack inspired by [7, 27, 10] 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 5. The safety properties is expressed in LTL syntax as

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

Figure 6 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

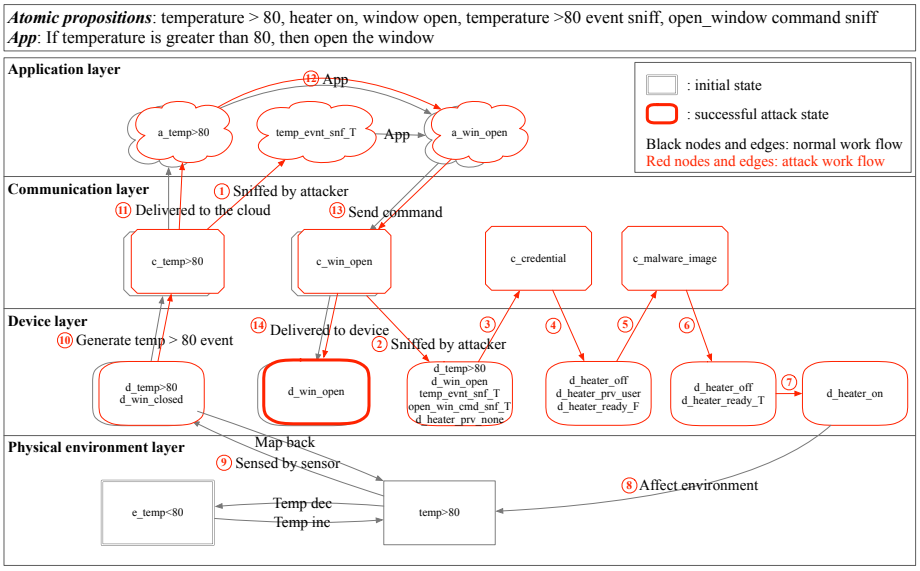

Figure 6: The hypothesis graph for a smart home with attack trace.

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 6.1. 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 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 6.

7.2. Smart Healthcare

Our framework can be easily applied to other IoT scenarios, such as smart healthcare, or smart factories. Here we show how to create IoT hypothesis graphs for an automatic blood glucose control system, where the insulin pump

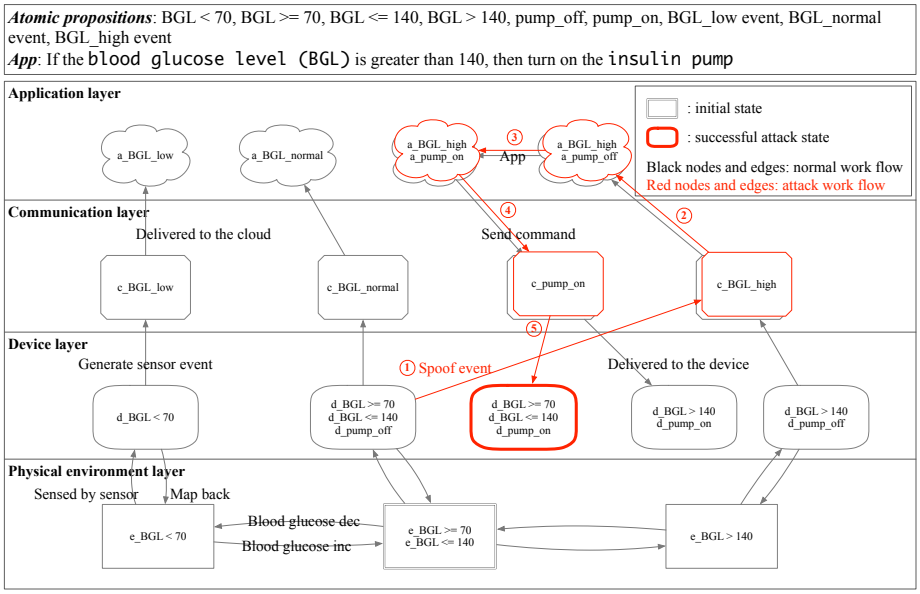

Figure 7: The hypothesis graph for an automatic blood glucose control system. The red shapes and lines illustrate the attack trace.

automatically injects insulin when the blood glucose gets higher the normal range. The correctness property is represented as the following LTL formula

$$\boldsymbol{G}(dev.pump.on \rightarrow dev.BGL.high).$$

The attacker's goal is to make the pump inject insulin when the user's glucose level is within the normal range, which may cause severe medical issues. Therefore, if we can validate the above correctness property, we will be able to decide whether the attacker can achieve his goal.

Different from smart home scenarios where the physical environment features are physical quantities such as temperature, illuminance, smoke, etc., in smart healthcare, the environment features are the patient's biological features such as blood glucose level, or blood oxygen saturation, etc. The devices are also medical sensors and actuators such as blood glucose sensor and insulin pump. The communication and application layer are similar to those of the smart homes,

representing communication events and decision maker states. The hypothesis graph of the above smart healthcare example is shown in Figure 7.

To differentiate atomic propositions at different layers, we add prefixes according to the layers they belong to. The atomic propositions are prefixed with `e`, `d`, `c`, `a`, from the lowest layer to the highest layer. Similar to Figure 6, to reduce clutter, we label each node with only the atomic propositions which hold true in that node.

The normal system transitions are represented in gray color, while the nodes and edges marked in red represent attack traces. Initially, the patient's blood glucose level (BGL) is within the normal range so the insulin pump is off. The attacker launches attack by first spoofing a BGL high event (as denoted by edge ①). Then the spoofed packet is delivered to the decision maker (②), triggering a command to turn on the pump (③). As a result, the pump is turned on even though the patient's actual BGL is normal (④, ⑤). Due to the initial attack edge ①, the resulting successful attack state can be reached from the initial state. Hence, the model checker detects the violation to the specified correctness property and returns the attack trace.

The above two examples show that since the four layers identified are essential for different IoT systems, and IoT cyber attacks also have similar mechanisms, our framework can be easily applied to different IoT scenarios.

8. Evaluation

8.1. Implementation

We implement ForeSee based on the Spin model checker [37]. Spin takes a system modeling language called Promela (Process Meta Language) [37] 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

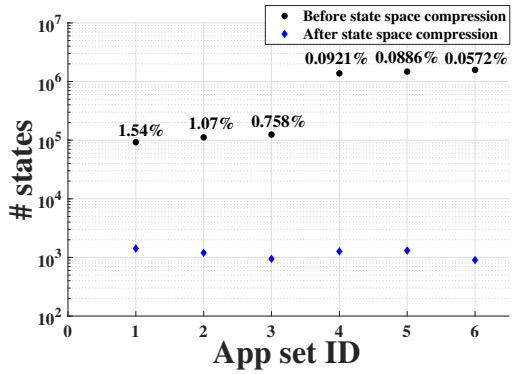

Figure 8: Number of states before and after compression and the compression ratio.

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.

8.2. State Compression Ratio

To evaluate the effectiveness of our dynamic selection algorithm, we created 6 smart home scenarios whose IoT apps are chosen from the Samsung SmartApps [38]. On average, each scenario consists of 7.4 apps. We count the number of states of the generated hypothesis graph before and after applying the dynamic selection algorithm. Compression ratios are computed by dividing the number of states of the hypothesis graph generated from the largest subsets of related apps by the one generated from the original app set. The results are shown in Figure 8, where the state compression ratios are shown above the black dots in the figure. We can see from the figure that the compression ratio ranges from 1.5% to 0.057%, and for the larger set of apps our algorithm tends to achieve better ratio.

8.3. 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

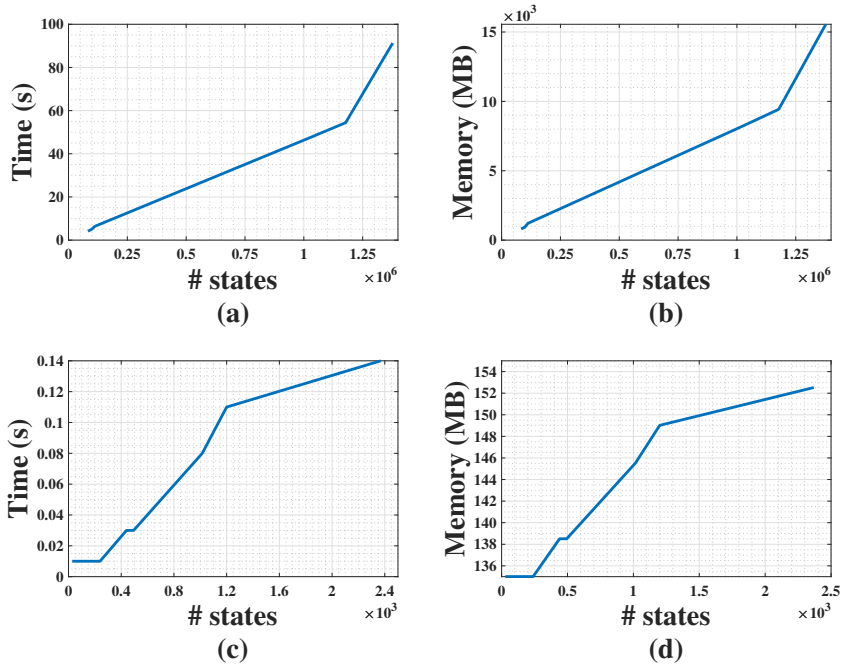

Figure 9: Impact of hypothesis graph size (measured by number of states) on verification time and memory usage. 9 (a) and 9 (b) show the scenarios where the models pass the verifier; 9 (c) and 9 (d) show the scenarios where there are violations to the property.

violation of the property. The time and space complexity of hypothesis graphs that pass the verifier are shown in Figure 9(a) and 9(b), while the ones of the hypothesis graph that fail to pass are shown in Figure 9(c) and 9(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 9 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 9. From the figures, we can see 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.

9. Related work

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

Device Layer. Costin *et al.* [39] conducted a static analysis of 32 thousand embedded firmware images and discovered 38 previously unknown vulnerabilities of embedded devices. Son *et al.* [19] presented real-world attacks to drones by employing the resonant frequencies of Micro-Electro-Mechanical Systems gyroscopes. Ronen *et al.* [40] 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.* [41] 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.* [42] 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.* [43] investigated the side-channel information leakage of video surveillance cameras through streaming traffic data analysis.

Application Layer. Ding *et al.* [27] 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.* [44] proposed a formal framework for IoT security analysis based on satisfiability modulo theories (SMT). [35, 45] took advantage of model checking to analyze application-level vulnerabilities in IoT systems. Mohsin *et al.* [46] presented a probabilistic model checking based framework to analyze the risks quantitatively.

10. Discussion

Our hypothesis graph uses model checking to detect IoT vulnerabilities. Researchers have also proposed other graph-based approaches for network security analysis [47, 48, 49]. In this section, we compare these methods and discuss benefits and limitations of our framework.

Cycles in the graph. Attack graphs created by [49, 47] utilize a key assumption of monotonicity, meaning once the attacker has gained some privilege, he will always have that privilege in the following attacks. However, this is not true for IoT systems because there could be some negative feedback loops due to automatic control. For example, when the attacker compromises the heater to increase room temperature, an IoT app will sense this physical event and automatically turn on the air conditioner to lower the temperature. Since attack graphs based on monotonicity assumption cannot deal with cycles, they cannot model and detect such attacks. In comparison, we do not make this assumption and our hypothesis graphs can uncover violations of correctness properties involving cycles.

State explosion. All of the model checked-based analyses encounter the state explosion issue. Even though our framework utilizes a dynamic selection algorithm, state explosion may still occur if the set of atomic propositions we need to consider gets larger. This can happen due to new IoT apps installed or devices enhanced with new capabilities, both of which introduce additional device interaction, connecting previously independent app sets. To further mitigate this issue, we select the bit-state hashing option during the verification. Instead of using the original state vectors, this approximation technique [50] uses the hash value of state vectors to index the state array, reducing memory usage to 1 percent while achieving the approximation ratio close to 100 percent[18].

11. 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. We also present an approach to extract the IoT application logic from app descriptions using NLP techniques, and show the experimental results. Besides, we design a state compression algorithm to reduce the size of the generated hypothesis graph. The framework can detect vulnerabilities and threats caused by any interaction between IoT core components. Our evaluation shows that the prototype scales well and works for hypothesis graphs with millions of nodes. The compression algorithm is able to reduce the number of states by three orders of magnitude.

12. 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] Top 10 iot applications in 2020, https://iot-analytics.com/top-10-iot-applications-in-2020/.

[2] Internet of things statistics, facts & predictions, https://review42.com/internet-of-things-stats/.

[3] Samsung, Smartthings, https://smartthings.developer.samsung.com.

[4] Google, Nest, https://developers.google.com/assistant/smarthome/ove\rview.

[5] IFTTT, Ifttt platform, https://platform.ifttt.com/.

[6] 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/.

[7] 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, Y. Zhou, Understanding the mirai botnet, in: 26th USENIX Security Symposium, 2017, pp. 1093–1110.
URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[8] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, P. Tague, Smartauth: User-centered authorization for the internet of things, in: 26th USENIX Security Symposium, 2017, pp. 361–378.
URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tian

[9] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, S. Unviersity, Contexlot: Towards providing contextual integrity to appified iot platforms., in: NDSS, Vol. 2, 2017, pp. 2–2.

[10] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, 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, 2018, pp. 1074–1088. doi:10.1145/3243734.3243820.
URL http://doi.acm.org/10.1145/3243734.3243820

[11] P. T. P. LLP, Z-shave. exploiting z-wave downgrade attacks, https://www.pentestpartners.com/security-blog/z-shave-exploiting-z-wave-downgrade-attacks/.

[12] P. T. P. LLP, Burning down the house with iot, https://www.pentestpartners.com/security-blog/burning-down-the-house-with-iot/.

[13] Y. Wan, K. Xu, G. Xue, F. Wang, Iotargos: A multi-layer security monitoring system for internet-of-things in smart homes, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 874–883.

[14] A. Wang, A. Mohaisen, S. Chen, Xlf: A cross-layer framework to secure the internet of things (iot), in: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2019, pp. 1830–1839.

[15] E. Fernandes, J. Jung, A. Prakash, Security analysis of emerging smart home applications, in: 2016 IEEE symposium on security and privacy (SP), IEEE, 2016, pp. 636–654.

[16] T. Sugawara, B. Cyr, S. Rampazzi, D. Genkin, K. Fu, Light commands: laser-based audio injection attacks on voice-controllable systems, in: 29th {USENIX} Security Symposium ({USENIX} Security 20), 2020, pp. 2631–2648.

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

[18] G. J. Holzmann, The model checker spin, IEEE Transactions on Software Engineering 23 (5) (1997) 279–295. doi:10.1109/32.588521.

[19] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, Y. Kim, Rocking drones with intentional sound noise on gyroscopic sensors, in:

24th USENIX Security Symposium, 2015, pp. 881–896.
URL https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/son

[20] NVD, Ring doorbell, https://nvd.nist.gov/vuln/detail/CVE-2015-4400.

[21] A. Cui, M. Costello, S. Stolfo, When firmware modifications attack: A case study of embedded exploitation, NDSS.

[22] NVD, Xiaomi zigbee devices, https://nvd.nist.gov/vuln/detail/CVE-2019-15913.

[23] NVD, August lock, https://nvd.nist.gov/vuln/detail/CVE-2019-17098.

[24] NVD, Eques elf smart plug, https://nvd.nist.gov/vuln/detail/CVE-2019-15745.

[25] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, P. McDaniel, Iotsan: Fortifying the safety of iot systems, in: Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, 2018, pp. 191–203.

[26] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, C. A. Gunter, Charting the attack surface of trigger-action iot platforms, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1439–1453.

[27] W. Ding, 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, 2018, pp. 832–846. doi:10.1145/3243734.3243865.
URL http://doi.acm.org/10.1145/3243734.3243865

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

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

[30] E. Y. Vasserman, N. Hopper, Vampire attacks: Draining life from wireless ad hoc sensor networks, IEEE Transactions on Mobile Computing 12 (2) (2013) 318–332. doi:10.1109/TMC.2011.274.

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

[32] G. J. Holzmann, W. S. Lieberman, Design and validation of computer protocols, Vol. 512, Prentice hall Englewood Cliffs, 1991.

[33] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. M. Wing, Automated generation and analysis of attack graphs, in: Proceedings 2002 IEEE Symposium on Security and Privacy, 2002, pp. 273–284. doi:10.1109/SECPRI.2002.1004377.

[34] Common vulnerabilities and exposures, https://cve.mitre.org/.

[35] Z. B. Celik, P. McDaniel, G. Tan, Soteria: Automated IoT safety and security analysis, in: 2018 USENIX Annual Technical Conference, 2018, pp. 147–158.
URL https://www.usenix.org/conference/atc18/presentation/celik

[36] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in neural information processing systems, 2013, pp. 3111–3119.

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

[38] Samsung, Smartthings public github repo, https://github.com/SmartThingsCommunity/SmartThingsPublic.

[39] A. Costin, J. Zaddach, A. Francillon, D. Balzarotti, A large-scale analysis of the security of embedded firmwares, in: 23rd USENIX Security Symposium, 2014, pp. 95–110.
URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin

[40] E. Ronen, A. Shamir, A. Weingarten, C. O'Flynn, Iot goes nuclear: Creating a zigbee chain reaction, in: 2017 IEEE Symposium on Security and Privacy, 2017, pp. 195–212. doi:10.1109/SP.2017.14.

[41] T. Gu, 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), 2018, pp. 254–262. doi:10.1109/MASS.2018.00047.

[42] Y. Jia, Y. Xiao, J. Yu, X. Cheng, Z. Liang, Z. Wan, A novel graph-based mechanism for identifying traffic vulnerabilities in smart home iot, in: IEEE INFOCOM, 2018, pp. 1493–1501. doi:10.1109/INFOCOM.2018.8486369.

[43] H. Li, Y. He, L. Sun, X. Cheng, J. Yu, Side-channel information leakage of encrypted video stream in video surveillance systems, in: IEEE INFOCOM, 2016, pp. 1–9. doi:10.1109/INFOCOM.2016.7524621.

[44] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, 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), 2016, pp. 180–188. doi:10.1109/CNS.2016.7860484.

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

10.1145/3281411.3281440.
URL http://doi.acm.org/10.1145/3281411.3281440

[46] M. Mohsin, M. U. Sardar, O. Hasan, Z. Anwar, IoTRiskAnalyzer: A probabilistic model checking based framework for formal risk analytics of the internet of things, IEEE Access 5 (2017) 5494–5505. doi:10.1109/ACCESS.2017.2696031.

[47] X. Ou, W. F. Boyer, M. A. McQueen, A scalable approach to attack graph generation, in: Proceedings of the 13th ACM conference on Computer and communications security, 2006, pp. 336–345.

[48] X. Ou, S. Govindavajhala, A. W. Appel, Mulval: A logic-based network security analyzer., in: USENIX security symposium, Vol. 8, Baltimore, MD, 2005, pp. 113–128.

[49] P. Ammann, D. Wijesekera, S. Kaushik, Scalable, graph-based network vulnerability analysis, in: Proceedings of the 9th ACM Conference on Computer and Communications Security, 2002, pp. 217–224.

[50] G. J. Holzmann, An improved protocol reachability analysis technique, Software: Practice and Experience 18 (2) (1988) 137–161.