Jumping the Air Gap: Modeling Cyber-Physical Attack Paths in the Internet-of-Things

Ioannis Agadakos*
Stevens Institute of Technology
Hoboken, New Jersey, USA
iagadako@stevens.edu

Chien-Ying Chen*
University of Illinois at Urbana-Champaign
Champaign, Illinois, USA
cchen140@illinois.edu

Matteo Campanelli*
The City College of New York
New York, New York, USA
mcampanelli@gradcenter.cuny.edu

Prashant Anantharaman*
Dartmouth College
Hanover, New Hampshire, USA
prashant.anantharaman.gr@dartmouth.edu

Monowar Hasan*
University of Illinois at Urbana-Champaign
Champaign, Illinois, USA
mhasan11@illinois.edu

Bogdan Copos*
SRI International
Menlo Park, California, USA
bogdan.copos@sri.com

Tancrède Lepoint
SRI International
New York, New York, USA
tancrede.lepoint@sri.com

Michael Locasto
SRI International
New York, New York, USA
michael.locasto@sri.com

Gabriela F. Ciocarlie
SRI International
New York, New York, USA
gabriela.ciocarlie@sri.com

Ulf Lindqvist
SRI International
Menlo Park, California, USA
ulf.lindqvist@sri.com

ABSTRACT

The proliferation of Internet-of-Things (IoT) devices within homes raises many security and privacy concerns. Recent headlines highlight the lack of effective security mechanisms in IoT devices. Security threats in IoT arise not only from vulnerabilities in individual devices but also from the composition of devices in unanticipated ways and the ability of devices to interact through both cyber and physical channels. Existing approaches provide methods for monitoring cyber interactions between devices but fail to consider possible physical interactions. To overcome this challenge, it is essential that security assessments of IoT networks take a holistic view of the network and treat it as a "system of systems", in which security is defined, not solely by the individual systems, but also by the interactions and trust dependencies between systems.

In this paper, we propose a way of modeling cyber and physical interactions between IoT devices of a given network. By verifying the cyber and physical interactions against user-defined policies, our model can identify unexpected chains of events that may be harmful. It can also be applied to determine the impact of the addition (or removal) of a device into an existing network with respect to dangerous device interactions.

We demonstrate the viability of our approach by instantiating our model using Alloy, a language and tool for relational models. In our evaluation, we considered three realistic IoT use cases and demonstrate that our model is capable of identifying potentially dangerous device interactions. We also measure the performance of our approach with respect to the CPU runtime and memory consumption of the Alloy model finder, and show that it is acceptable for smart-home IoT networks.

*These authors contributed equally to the paper. This work was performed at SRI International's Internet of Things Security and Privacy Center. Ioannis Agadakos, Chien-Ying Chen, Matteo Campanelli, Prashant Anantharaman and Monowar Hasan were at SRI International when this work was performed.

CPS-SPC'17, November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-5394-6/17/11…$15.00
https://doi.org/10.1145/3140241.3140252

1 INTRODUCTION

The ease of adoption, quick setup, and autonomy makes it easy for the layman to integrate IoT devices into their everyday life; a smart personal assistant, that can interact with a variety of devices including smart light bulbs, smart outlets, and smart watches, can be installed in a matter of minutes. However, the characteristics that make IoT so powerful and useful (i.e., interoperability, autonomy, and their cyber-physical nature) also introduce security and privacy concerns. In recent years, cybersecurity researchers have warned about the implications and risks of the Internet-of-Things (IoT) [1, 6, 11, 17, 25].

Device autonomy eliminates the need for extensive user configuration. Interoperability enables devices to impact each other directly,

via network communications, or indirectly, through the physical environment. These characteristics in combination with the complex and ad hoc nature of the device interactions can lead to surprising behaviors that defy expectations. For example, a Honeywell cyber-security researcher recently demonstrated how a seemingly innocent command to a valve of a industrial pump system can indirectly, through air bubbles, cause a sensor to decrease the efficiency of the pump and ultimately stop it.[1] This risk is aggravated by the dynamic nature of IoT systems and their ability to impact the physical environment. Over time, users introduce (and remove) devices into their IoT networks with no way of determining the impact on the security and privacy of such changes. The introduction of a new device in an existing network can provide a new window of opportunities for third parties to abuse and attack the users, elicit sensitive information, disrupt assets [1], or directly impact the physical world.

Our team of researchers studies the idiosyncrasies of IoT and the devices that comprise it. We recognize that the cyber and physical natures of these devices cannot be separated. Consequently, we argue that tools for security assessment in IoT systems should be able to explicitly express the interactions between heterogeneous devices and account for physical interactions because they:

- Can create additional attack vectors;
- Are independent of cyber interactions among devices (e.g., Wi-Fi or a TCP connection), the usual focus of network security assessments;
- Are difficult to detect;
- Are difficult to harden.

To capture these dynamics, we believe a model should follow a "pessimistic assumption" (that yet proves to capture realistic attacks, cf. Section 5), where device interactions are defined by the (hardware) capabilities of the devices and are not limited by the software stack driving the devices and their interactions. In other words, for a particular device with a light sensor, HVAC controlling capabilities, and a Bluetooth radio, we consider the possibility for the light sensor to trigger the HVAC controls even if the software of the device does not define such behavior. This pessimistic perspective enables us to reason about attackers of various capabilities and consider worst-case scenarios.

To validate our approach, we built a prototype system using Alloy [9], a state-of-the-art model analyzer, that instantiates our "pessimistic assumption" model. We consider three use cases to evaluate our approach and demonstrate how it can foresee and warn against realistic attacks while offering valuable insight on what enabled the attacks in the first place.

The main contributions of our research are as follows:

(1) A way of modeling cyber and physical relations between devices in an IoT network;
(2) A way of revealing potential attack paths from both cyber and physical interactions within a given network;
(3) A way of enumerating the prerequisites, including device states, that make a particular attack path possible.

[1] https://www.wired.com/story/evil-bubbles-industrial-pump-hack/

2 BACKGROUND AND RELATED WORK

2.1 Internet-of-Things

While IoT devices are similar in nature to cyber-physical systems, some characteristics are unique to this class of devices.

Software Components. Interactions between devices can change often and are driven by software external to the devices. For example, in the Samsung SmartThings environment, the interactions between devices are defined by the SmartApps. These applications can be added and removed by users over time. The addition (or removal) of new devices can also introduce changes in the interactions between devices.

Channel Projection. The closely intertwined software and hardware grant IoT systems a unique ability to seamlessly project from the physical world to the cyber world. A device, such as a smart thermostat, can receive commands from a smartphone, which can trigger actions that impact the physical environment (e.g., turn the AC on).

Ad hoc Interoperability. IoT devices can affect each other in ad-hoc ways, and can be configured in unforeseen setups. Interoperability implies that the security of a device is defined by its peers and the interactions between devices. For example, when a smart electric plug is installed in series with a SmartThings hub, even if the SmartThings hub is properly secured, its behavior can be influenced by the insecure smart plug. More concisely, vulnerabilities can be introduced by composition of devices, regardless of the security of each individual device.

2.2 The Alloy Analyzer

To capture the complicated interactions between devices, we propose a way of modeling and an implementation using *Alloy* [9]. Alloy is a simple, expressive language for describing complex structures. Paired with a model checker, Alloy can be used to model various systems and verify properties of the model. Alloy has been previously used to model "all the possible security configurations of a web application, or all the possible topologies of a switching network". [2]

We choose Alloy due to its balance of expressiveness and ease of use making it an ideal candidate for describing the capabilities of IoT devices and the complexity of their interactions. One of the goals of this work is to search all possible combinations of interactions and find a counterexample, if one exists, for a particular property, a feature support by the Alloy analyzer.

The Alloy analyzer works by reduction to SAT (an NP-complete problem). While scaling to large input sets is generally challenging for a satisfiability solver, the scope of our application is limited to smart home environments, comprised of a finite set of a few dozen devices, which along with careful modeling enables reasonable performance time (see Section 5.4). Although a complete Alloy tutorial [3] is beyond the scope of this paper, we recall below features that are critical to our work.

[2] Entire text can be found at http://alloy.mit.edu/alloy/.
[3] We refer the interested reader to Alloy official tutorial: http://alloy.mit.edu/alloy/tutorials/online/.

Signatures. In Alloy, an entity can be abstract or concrete, which enables inheritance-like abilities. For example, consider the following snippet of an entity definition, `Device`, which can be either `Actuator` or `Sensor`. This simple example highlights the subtle but expressive power of Alloy.

```
abstract sig Device{}
sig Sensor   extends Device{}
sig Actuator extends Device{}
```

Facts. Facts are universal truths about how things operate in the universe we describe. For example, the following statement indicates that devices can either be `ON` or `OFF`. The use of implications guarantees that it is not possible for a device to be simultaneously `ON` and `OFF`. It is also possible to define symmetric relationships, as shown with the definition of `adjacency`. This means that if location A is adjacent to location B, then location B is adjacent to location A.

```
fact noQuantumStates{
  all d:Device,s:State |  ON->d in s.on/off iff OFF->d not in s.on
      ↪ /off
  all d:Device,s:State | OFF->d in s.on/off iff  ON->d not in s.on
      ↪ /off
}
fact adjacency{
  adjacent=~adjacent
}
```

Assertions. If facts can be seen as axioms that are always true, then assertions can be thought of as theorems that should be provably true based on the supplied facts and the provided inputs. If an assertion is not true, the model checker searches for counterexamples. Alloy enables users to define assertions of various complexity, from simple sanity/policy checks to more complicated logic scenarios that could reveal an attack. The following code snippet shows a simple assertion that checks whether devices with no authentication capabilities are in publicly accessible areas:

```
assert noPublicAccess{
  no r:Location,d:r.devices,channel:d.channel | r.access=Public ∧
      ↪ (some d.authentication & channel->NoAuth)
}
```

Predicate. Predicates are statements that, like assertions, are verified given the facts and inputs provided. In contrast to assertions, predicates provide a possible solution as derived from the model. The following predicate example produces a network graph based on the inputs and facts defined in our model:

```
pred ShowNetwork{}
```

Figure 3a shows a possible outcome from this predicate.

2.3 Security in IoT

Modeling systems and their behavior has been extensively studied. Researchers have demonstrated the use of languages [23] to describe systems and how such descriptions can be leveraged to formally verify (security) properties of the systems.

Significant work has also been done in threat modeling. Threat modeling aims to analyze the security of a system and identify potential threats by approaching the problem from the perspective of a hypothetical attacker. A popular method for threat modeling is by leveraging attack trees [20]. Attack trees represent attacks and countermeasures as a tree structure, where the root node represents the goal of the attack and the rest of the nodes denote attacker moves. Other efforts leverage attack graphs to evaluate the security of various systems [12, 16, 24], while others study methods for automatically generating them [18, 21].

More recently, researchers have begun applying some of the previously mentioned methods to cyber-physical systems and specifically IoT systems. Kovatsch et al. [10] proposed a technique to build a web-like mesh of IoT devices by composing the representational state transfer (REST) based APIs of various IoT devices using a rule-based reasoning engine. In their work, the authors found that their model led to a state-space explosion for large networks. Unlike previous work [10] that focused only on the application layer, we use a *satisfiability-based approach* and look at the protocols from both the data-link as well as the application layer. Mohsin et al. [15] proposed IoTSAT, a formal framework for the security analysis of IoT networks. They also follow a satisfiability-based approach to model and analyze security. These papers, however, make different assumptions in their respective models. First, IoTSAT assumes that an attack on actuators can affect sensors only through the sensors' observations of the environment [15, Figure 3]. We do not make such assumptions (i.e., interactions of only type and direction: Sensors → Controllers → Actuators), thus accounting for arbitrary ways in which actuators can affect sensors (or any other types of devices). This allows our model to detect attacks undetectable in IoTSAT, such as the one described in Section 5.1. Additionally, we do consider controllers and actuators as potential starting points of attacks and do not assume them to be adequately hardened.

An event condition action (ECA) language-based approach was used by Guilly et al. [7] to extend ECA using timed automata by applying it to arrest faults and unsafe conditions in a home automation system. Corno and Sanaullah [3] proposed a design-time modeling and formal verification methodology for smart environments using ontologies and state charts. Augusto and Hornos [2] presented a methodological guide on use of linear temporal logic (LTL) to model, simulate, and verify intelligent environments. Coronato and Pietro [4] propose guidelines for requirement specifications and verification through extensions to ambient calculus and ambient logic. All of these research efforts are focused on verifying the inherent and non-malicious behavior of specific systems and lack generality and security analysis against different attack vectors on such environments that distinguish the proposed method.

Non-formal techniques have also been applied to evaluate and enforce the security and privacy of IoT systems. IoT Sentinel [14] is a system that uses software-defined networks to restrict the communications of devices according to a set of predefined policies. Both IoT Sentinel and our work perform network traffic analysis to automatically identify devices on a given IoT network. Our work, however, aims to determine potential attack paths while IoT Sentinel focuses on mitigation and isolation techniques. Mavropoulos et al. [13] developed a tool named ASTo which uses a domain-specific modeling language to generate a visualization of IoT systems, which facilitates security analysis during the design and implementation phases. Rullo et al. [19] applied a game-theoretic approach to IoT security modeling to allow a defender to efficiently allocate resources to protect an IoT system. Some limitations of their work are abstracting away the heterogeneity of IoT system and lacking system-level evaluation. Tekeoglu et al. [22] proposed a testbed for

the evaluating the security and privacy of IoT systems by analyzing layer 2 and 3 packets. Although our work also analyzes network traffic, we go beyond observed network traffic to draw conclusions about the security and privacy of an IoT system. Geneiatakis et al. [5] studied the interactions between devices in a real smart-home test-bed and used the information to make inferences about general IoT security and privacy flaws, while considering both internal and external attackers with varying capabilities. While the output is similar, our approach does not involve manual analysis or speculation. Our approach automatically verifies security properties of an IoT system automatically through model checking.

3 SYSTEM MODEL AND DESIGN

Our research focuses on the interactions between devices in an IoT network. Typical assessments of system interactions consider only network connections established by wired or wireless channels; however, both cyber and physical aspects play an important role in IoT networks and are often unanticipated.

In this section, we introduce a way of modeling an IoT network that captures both cyber and physical relations between devices. While there is a large variety of attributes that identify a device (e.g., color, casing material, etc.), for our purposes we focus only on attributes that affect and enable interactions between devices. For instance, we consider I/O (physical or cyber) capabilities and location, among other features. The properties that comprise our model are described along with snippets of code portraying our current Alloy implementation.

3.1 Device

A *device* is an instance of an electronic device deployed in an IoT network. It can be as simple as a sensing device (e.g., temperature sensor, light sensor), an actuator (e.g., thermostat, oven) or as sophisticated as a commodity computer (e.g., laptop, IoT hub). Each device is characterized by a physical location and supported input/output channels, as defined by the device hardware capabilities. Device definitions are generated manually. Information regarding device capabilities, including protocols supported, and sensing and actuating abilities, can be inferred from device documentation available on manufacturers' websites. Our current approach uses sniffers to identify device types and uses manually generated device definitions to identify capabilities (e.g., I/O channels).

Our model does not differentiate between device roles (i.e. sensors, coordinators, and actuators). This allows us to capture all possible cyber and physical relations between devices. While the software controlling the devices may forbid some interactions, treating all devices (and roles) equally provides a more general way to see the potential attack paths and realize the value gained by comparing a worst-case scenario to a more realistic approach. Other devices inherit from this abstract model and set specific values for each attribute; an example model of a Philips TV is shown below.

```
abstract sig Device{
  access:                lone Access,
  //some devices have access rights and are able to modify other
      ↪ devices or perform tasks
  channel:               some Channel,
  authentication:        Channel->Authentication,
  defaultAuthentication: Channel->Authentication,
  size:                  one Space,
  connections:           Channel->Device,
  inChannel:             some Channel,
  subspace:              lone Location,
  mobility:              one Mobility,
  power:                 one Power,
  defaultChannels,defaultInChannels,defaultOutChannels: set
      ↪ Channel,
  outChannel:            some Channel
}{
  defaultChannels       = Heat + PowerAC + KeySwitch
  defaultInChannels     = PowerAC + KeySwitch
  defaultOutChannels    = Heat
  defaultAuthentication = KeySwitch->NoAuth + PowerAC->NoAuth +
      ↪ Heat->NoAuth

}
abstract sig Authentication {}
one sig StrongAuth,WeakAuth,NoAuth extends Authentication{}
```

Listing 1: While the Alloy Language looks very similar to object oriented Java, one should not mistake the two and follow object oriented practices blindly; the purpose of Alloy is to define a model and allow for formal validation, albeit in an expressive way. For coherency, elements and relationships prevalent in many devices are placed in the "default" sets and are inherited to every device. Ad-hoc devices that deviate from these default sets must have any extra default elements explicitly removed during their definition.

```
abstract sig PhilipsTV extends Device{}
{
  authentication = defaultAuthentication+WiFi->StrongAuth + Voice-
      ↪ >NoAuth
  channel        = defaultChannels + WiFi + Voice
  inChannel      = defaultInChannels + WiFi
  outChannel     = defaultOutChannels + WiFi + Voice
  size           = small
  power          = PowerWire
  mobility       = Immobile
}
```

Listing 2: An example device instance. This Phillips TV inherits all default channels and also defines any extra capabilities this device has, i.e the ability to produce output in the Voice and WiFi channels; notice that this particular model cannot receive input in the Voice channel (does not have a microphone).

3.2 Channel

A device can have multiple channels via its input and output interfaces that enable interactions. Our model considers both cyber and physical channels. For example, Wi-Fi, Bluetooth, ZigBee, and Z-Wave are typical examples of cyber channels used in IoT. Temperature, humidity, voice, and vibration are examples of physical channels utilized by IoT devices. The channels can be inferred from device documentation and capabilities.

Our model also defines constraints on the interactions based on the physical proximity of devices (further elaborated in Sections 3.4 and 3.5). Devices having a common interface and located within the effective range can interact with each other. For example, a Bluetooth-enabled lock can interact with a smartphone via the Bluetooth channel when they are within Bluetooth signal range. Similarly, an air conditioning unit producing cold air cools down room temperature, which is sensed by a temperature sensor in

the same or adjacent room. A temperature channel connects the air conditioning unit and the temperature sensor. In contrast, a smart light has no such a temperature channel connected with the air conditioning unit as its light sensor is not influenced by the AC. A device cannot have a channel with any other device if the corresponding interface is not supported. For instance, Amazon Echo can not establish a ZigBee channel with any other device because Amazon Echo does not support ZigBee. The following listing describes how channels are modeled in Alloy.

```
abstract sig PHY {} // PHY stands for "physical layer"
abstract sig Channel {
        medium: one PHY, range: one Space }
// only some mediums are shown for demonstration purposes
one sig Movement,RF_802_15_4,Acoustic extends PHY{}
one sig ZigBee extends Channel{}{
        medium=RF_802_15_4 range=large}
//although Zigbee would require two different PHY normally
//since it can be 2.4Ghz  or  900Mhz
//we model the 2.4Ghz version here
```

Listing 3: We model as "Channel" anything that may enable interaction between devices. A principal component of a Channel is the "range", within which such an interaction is effective under normal operation.

3.3 Elements of Security

Our model also allows us to capture various notions of security. Since our objective is to identify potentially dangerous (sets of) device interactions, we focus on authentication, although confidentiality and integrity can be similarly modeled. A device that communicates with other devices via unauthenticated channels may potentially serve as a stepping stone to an otherwise properly configured network. Our model defines three authentication levels for a given channel: strong-authentication, weak-authentication or no-authentication.

Strong-authentication represents a channel that is usually considered hard to attack. For example, a Wi-Fi channel with WPA2-PSK enabled is considered to have strong-authentication. Weak-authentication suggests that a channel has some degree of authentication, but it is likely to be compromised by a smart attacker. For example, Apple's intelligent personal assistant, Siri, can be trained to recognize particular voices and only receive channels when the voice matches. However, a smart attacker could record the user's voice and activate Siri unexpectedly. No-authentication indicates that no authentication mechanism is applied to the given channel. Most physical channels are unauthenticated. For example, the temperature channel between an AC and a thermostat is not enforced by any authentication methods. We model authentication as a unidirectional relationship between Channel and Authentication and define it per device, as shown in listing 1. This approach allows us to express and define each device's capabilities in a fine grained manner.

3.4 Location and Accessibility

In many IoT settings, the nature of the physical location of devices can be important since it defines what (types) of entities have access to such locations and, consequently, to the devices. For example, electronic personal assistants, such as Amazon's Echo, should not be placed close to windows or outside walls, as they can pick up voice commands from individuals outside the building. Such notions are captured in the Location and Access entities shown in listing /reflocals.

In our approach, we define the following Access entities: public, private, and protected. More types can be defined, as needed. We also assume a hierarchical order of the Access types. For example, someone who can access a Private area can also access a Protected area and Public areas. When applying the model, the user must manually define location attributes for every physical location. Users must also define the location of each device (e.g., thermostat is in Living Room)

Public-access locations are accessible by the general public (such as doorway, backyard and hallway). A device deployed in a public location without protection is susceptible to attackers within reach. Private-access locations are considered well protected (or isolated from outsiders) such as bedrooms, private garages and document storage rooms in office buildings. Protected-access locations are locations such as the common areas in an apartment building protected by a doorman, where despite screening processes, devices are at risk in the presence of malicious visitors.

```
open util/ordering[Access]
abstract sig Location {
  access:        set Access,
  size:          one Space,
  sublocations:  set Location,
  root:          lone Location,
  distance:      Location->Space,
  adjacent:      some Location
}
abstract sig Access{}
one sig Public extends Access{}
one sig Private extends Access{}
one sig Protected extends Access{}
fact accessRights {
//imposing ordering on access rights
  lt[Protected, Public] ∧ lt[Private, Protected]
}
```

3.5 Proximity

Location provides high-level information that can be used to determine the accessibility of devices by various entities but, as defined, it only provides partial information regarding the possibility of interaction between any two particular devices. To combat this challenge, our model considers proximity. For example, some interactions types, such as Bluetooth communications, require near-proximity while others, such as interactions via sound, do not. For example, a Bluetooth-equipped device in a Private location could be susceptible to attacks depending on the range of Bluetooth and the capabilities of the attacker's hardware. This is captured in our model by employing a *fact*, which states that a device (or attacker) can only interact with another device through a channel if it is within the range defined by the channel. Our second case study 5.2 demonstrates the use of the Proximity rule. Specifically, it shows that an attacker is capable of interacting with an insecure device over the BLE channel due to the range of BLE, despite being unable to enter the Living Room location. Proximity information is defined manually per channel type.

3.6 Time and States

Our model supports scenario predictions by modeling time as a state where volatile associations of atoms are stored. This allows us to capture passage of time as a sequence of actions that transition a network of things from state S to S', consequently enabling the model to account for potential future changes in the topology of the network. For example, the following Alloy snippet shows how a state can be used to describe whether a device is *ON* or *OFF*. It follows that a Switch action is a function that changes the PowerState of a Device from S to S', a Move is a function that changes the Location of a Device from A to B, between two states, and so on.

```
sig State {
  on/off:    set PowerState->Device,
  location:  set Location->Device
}
pred switchOff[s,s': State,d: Device]{
  ON->d  in s.on/off iff s'.on/off-s.on/off = OFF->d
}
pred switchOn[s,s': State,d: Device]{
  OFF->d in s.on/off iff s'.on/off-s.on/off = ON->d
}
```

3.7 Attacker's Capabilities

Our model focuses on identifying the potential attack paths from the cyber and physical interactions between devices. When assessing different scenarios, it is important to consider the risk posed by attackers on a per-device level. This can narrow down the number of attack paths according to the environment of each use case. For example, some scenarios may only consider attackers with access to public locations, while others may consider the insider attacks, where the attacker has access to any area. In our approach, we model the attacker as an independent entity, identical to other entities, as shown in listing 3.7. An attacker with the "attackCapability" of weak authentication and public access can compromise anything reachable from public areas that has weak authentication.

```
abstract sig Attacker{
  attackCapability: some Authentication,
  access:           some Access
}
sig weakAttacker extends Attacker{}{
  // can be one of NoAuth|WeakAuth|StrongAuth
  attackCapability = NoAuth
  access           = Public
}
```

4 IMPLEMENTATION

Our system consists of two parts: (1) a set of live sniffers that can detect and identify devices that comprise an IoT network and (2) a model checker implemented in Alloy. The structure of the prototype system is shown in Figure 1. In our current implementation, we manually convert the captured device information into the Alloy model.

As discussed in Section 2.2, we chose to implement our model in Alloy for its expressiveness (Alloy supports first-order logic) and for its object-oriented programming flavor. For a more detailed discussion of the expressiveness and performance of our Alloy implementation see Section 5.

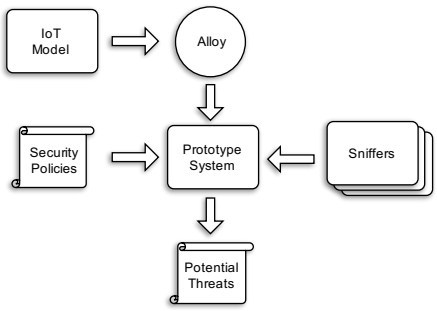


Figure 1: The high-level structural view of the implemented prototype system. The proposed model is realized with Alloy. The sniffers detect and update necessary device information in real-time. The system checks the constraints to detect potential attack paths in the given IoT network.

4.1 Sniffers

The sniffers provide real-time information about devices available in a particular IoT network. This allows for our approach to be dynamic in nature. In other words, the sniffers enable the model to be updated in real-time as changes in the network occur.

The sniffers passively monitor network transmissions between devices across several network technologies (e.g., Wi-Fi, Bluetooth, ZigBee) and many standardized protocols. The observed traffic is then analyzed and device fingerprinting is attempted. Our sniffers also take into account the mobility of IoT devices and as such, associate "presence timers" with each device. If no communications originate from a particular device within the duration of the "presence timer", the device is marked as "away" and removed from the current representation of the network topology. We do not guarantee completeness in the fingerprinting of devices using our sniffers. Although currently our sniffers rely on passive network monitoring, in the future we wish to improve our sniffers' capabilities and accuracy. Device fingerprinting is still an open research problem and while relevant to our overall problem, it is outside the scope of this paper.

The inferred device information is stored in a database that provides a centralized place for the system to read the runtime data and populate the model for the model checker (Figure 2). This design enables scalability and robustness by easily supporting additional sniffers.

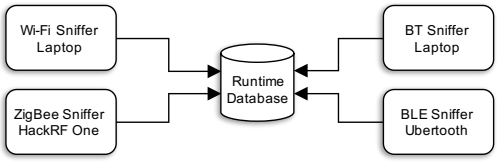


Figure 2: Device information is captured and identified by the sniffers and pushed into our database. Four types of sniffers are employed: Bluetooth (BT) sniffers, Bluetooth low-energy (BLE) sniffers, Wi-Fi sniffers, and ZigBee sniffers.

We developed the following sniffers:

(1) BT and BLE Sniffers:
We use Ubertooth[4], an open source Bluetooth monitoring and development platform, and the Bluetooth Linux library BlueZ[5] to scan advertisement packets from nearby devices. Then we use a self-modified version of BlueZ to automatically go through the list of detected MAC addresses and retrieve their primary characteristics.
(2) Wi-Fi Sniffers:
We use laptops as Wi-Fi sniffers and develop a Python program using a Python wrapper for Wireshark[6][7] to detect nearby Wi-Fi connections pairs.
(3) ZigBee Sniffers:
We use the HackRF One[8], a software-defined radio, to build a ZigBee sniffer. We create a IEEE 802.15.4 transceiver revised from an existing library[9] in GNU Radio. The output from the IEEE 802.15.4 transceiver in GNU Radio is a data stream that outputs the captured raw packets (the packets that contain IEEE 802.15.4 headers). A Python program is developed to receive the packet stream, extract device information (PAN ID, source, and destination IDs) and push the processed data to our database.

4.2 Model Checker (Alloy)

We use Alloy 4.2 and SAT4J as the SAT solver. Our prototype sniffers create simple signatures based on detected devices of the following form:

```
one sig PhilipsTV_1 extends PhilipsTV{}
{LivingRoom->this in ord/first.position}
one sig AmazonEcho_1 extends AmazonEcho{}
{LivingRoom->this in ord/first.position}
```

Device locations and additional connections that cannot be detected must be provided manually by the user as input.

5 EVALUATION

We use three case studies using real-world device setups to evaluate the efficacy of our approach. We choose attack scenarios using devices and configurations that would likely be encountered in a smart-home setting. Our testbed comprised the following devices:

- Amazon Echo
- OORT Smartplug (a BLE-enabled AC power plug)
- Samsung SmartThings Hub (a coordinator for the SmartThings protocol that operates on top of ZigBee)
- Samsung Multipurpose Sensor (a sensor used to detect a window's open/close status)
- Phillips SmartTV
- Kevo Smartlock (a Wi-Fi-enabled door lock)

[4]Ubertooth official website: http://ubertooth.sourceforge.net/
[5]BlueZ official website: http://www.bluez.org/
[6]Wireshark Python library: https://pypi.python.org/pypi/pyshark
[7]Wireshark official website: https://www.wireshark.org/
[8]HackRF One official website: https://greatscottgadgets.com/hackrf/
[9]IEEE 802.15.4 O-QPSK transceiver for GNU Radio: https://github.com/bastibl/gr-ieee802-15-4

5.1 Use Case: Hidden Paths

As physical channels are usually overlooked, paths partially or fully constructed by physical channels are hidden until they are discovered. In this use case, we show that a simple IoT home setup can contain potentially dangerous hidden paths. In a scenario with an Amazon Echo and a Philips TV, the Amazon Echo is connected to a Kevo smart lock, and all three devices are connected to a Wi-Fi network. The expected interactions between devices and the setup are shown in Figure 3c. The input to our system is:

```
one sig PhilipsTV_1 extends PhilipsTV{}
{ location=LivingRoom }
one sig AmazonEcho_1 extends AmazonEcho{}
{ location=LivingRoom }
one sig WiFiAP_1 extends WiFiAP{}
{ location=LivingRoom }
one sig KevoLock_1 extends KevoSmartLock{}
{ location=LivingRoom }
```

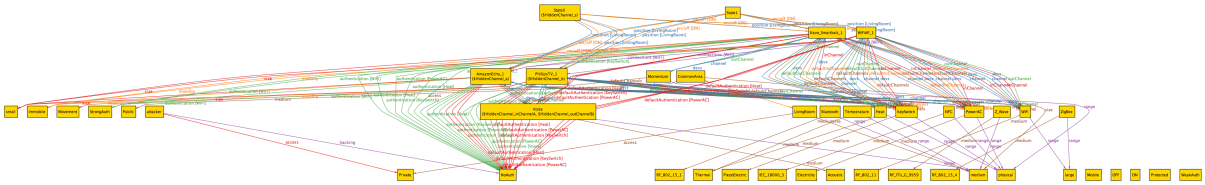
Our system generates every implicit connection and dependency based on the facts provided as input. Explicit connections are also used but are not required. Our approach does not assume that a communication link is possible only because we explicitly added an edge; thus, if a device receives input from a given channel, it can potentially be reached by any device producing output in that channel. Figure 3a shows the holistic view of the network produced by the Alloy implementation. The model exposes an unexpected potential attack that takes advantage of an illegal communication occurring between the Philips TV and the Amazon Echo over the unprotected voice channel. Moreover, the model asserts that the TV can provide output to the voice channel, while the Amazon Echo receives input without any authentication between the entities. The feasibility of this attack is left to the user/administrator to assess; the goal of our system is to expose potential hazards, provide the user with detailed information and enable informed-decision making. Recent news have provided examples of how such unexpected device interactions over hidden channels can lead to asset compromise.[10]

5.2 Use Case: Security Degradation

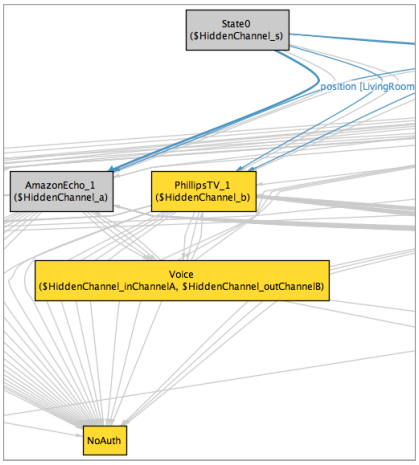
Our second scenario highlights the potential hazards of device composition. We show this by adding a security constraint to our implementation and allowing Alloy to find any violations of the constraint. In this example, we want to guarantee that any input only passes through authenticated channels. This dangerous scenario is analogous to the *confused deputy problem* [8] in that any node accepting input over unauthenticated channels but whose output is communicated over authenticated channels can be fooled to misuse its authority. If this property holds, the system shows no security degradation.

This experiment assumes an IoT smart home setup composed of an open/closed sensor, a SmartThings hub, and a smart plug. The sensor is installed on a window and triggers user alerts when the window is opened. The communication between the sensor and the SmartThings hub is protected by AES-128 bit encryption. The OORT smart plug is connected to a multi-extension cord that, with amongst other devices, connects to the SmartThings hub. The smart

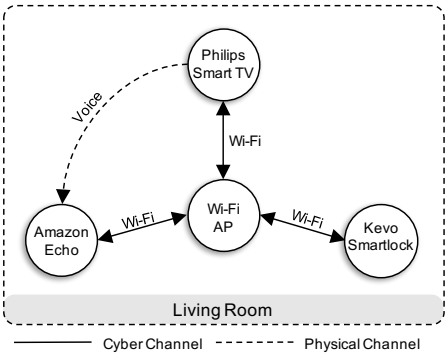
[10]TV unexpectedly interacts with Amazon Echo: https://www.theverge.com/2017/1/7/14200210/amazon-alexa-tech-news-anchor-order-dollhouse



(a) Our model follows a pessimistic approach; a device that receives input from a given channel may potentially receive input from any device that can produce output in that channel.



(b) Detail of the output visualization from the Alloy Analyzer. A counterexample was found at State0 that could lead to an attack. The main actors in the provided counterexample are the TV and Amazon Echo, and the assertion is falsified over the Voice channel.



(c) A topological map for the use case in a small home IoT setup. Circles are IoT instances of IoT devices, the solid lines represent the cyber channels, and the dashed line represents the hidden channel. The analysis indicates a potential attack path over a hidden, physical channel: Phillips smart TV → Amazon Echo → Wi-Fi AP → Kevo smart lock.

Figure 3: Our use cases as models found by the Alloy Analyzer and as topological maps.

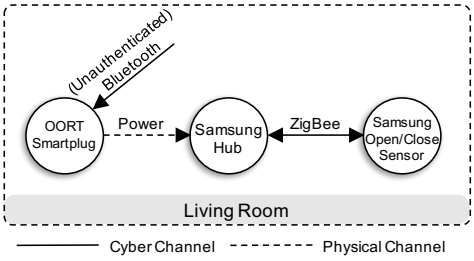


Figure 4: A topological map for studying security degradation. Representations are described in Figure 3c. In this use case, an OORT smart plug has an unauthenticated Bluetooth (anyone can connect to its Bluetooth), which introduces an attack path that compromises other devices that support stronger authentication.

plug is equipped with a BLE interface used for user control actions such as turning it on/off and extracting historical logs regarding power consumption. The BLE communications to the plug are not authenticated. The following Alloy model instance captures this scenario:

```
one sig LivingRoom extends Location{}{
  access = Private
  size   = medium
}
one sig SmartThingsHubSmartplug extends OORTSmartplug{}{
      ↪ location = LivingRoom
   connections = PowerAC->SmartThingsHub_1
   authentication=BLE->NoAuth
}
one sig SmartThingsHub_1 extends SmartThingsHub{}{
        location = LivingRoom
}
one sig SamsungMultiSensor_1 extends MotionSensor{}{
        location = LivingRoom connections = ZigBee->
              ↪ SmartThingsHub_1
}
```

Using this description of the environment as input, the model checker warns of a security degradation violation, as shown in Figure 4. The smart plug can affect the SmartThings hub via the power (physical) channel. Since the BLE channel to the smart plug is unauthenticated, an attacker could use spoofing commands to turn off the smart hub, consequently making the window sensor useless.

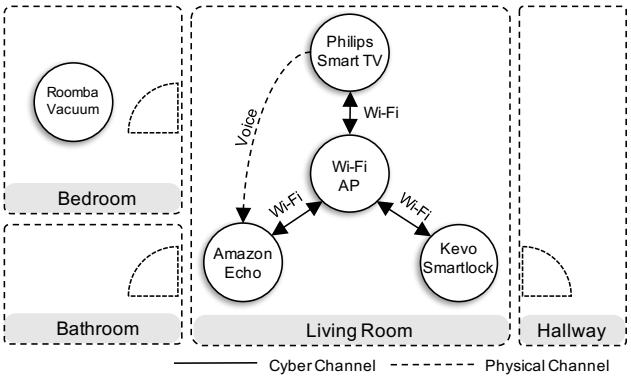


Figure 5: A topological map for a home IoT setup with multiple rooms. Representations are described in Figure 3c. In the initial state, a Roomba vacuum is located in the bedroom, designated `Private` location.

Table 1: A counterexample to the assertion, "Roomba can never be in a Public Access area at any time." Alloy does not provide or guarantee the same result for every run, it only guarantees that a solution will be returned from the set of possible counterexamples.

State	Roomba	NaiveLock	Transition
S0	Bedroom	Locked	Initial State
S1	Living Room	Locked	LivingRoom → Roomba
S2	Living Room	Unlocked	NaiveLock → Unlocked
S3	Hallway	Unlocked	Hallway → Roomba

5.3 Use Case: Transitions and States

The use cases examined so far, while based on real setups, are static, i.e., our system was finding ad hoc violations or hazards over a single state. By modeling transitions and states, our system can capture more sophisticated violations that span over time[11] and are composed of a sequence of discrete steps. This approach allows us to find counterexamples that violate an assertion without following a heuristics-based algorithm. All solutions are based entirely on the provided facts and explores the entire space of possible solutions. If a counterexample is not found, Alloy guarantees[12] that there is no counterexample up to the provided search space, but does not guarantee that no solution exists for a larger number of atoms.

To demonstrate our system's ability to capture these Movement and State transitions, we consider a Roomba vacuum, which can move. The setup in Figure 5 includes the bedroom, bathroom and living room as private locations and the hallway as a public location. The hallway, while physically adjacent and accessible to the private spaces, is isolated by a smart lock that can (like the OORT smart lock) be compromised over the BLE channel. This smart lock, when unlocked, opens a garage door. In the initial State S_0, the Roomba vacuum is located in the bedroom, a private location. With this scenario as input to our Alloy prototype, we attempt to verify the assertion: "Roomba can never be in a public access area, at any time." The assertion fails and a counter-example is found in as few as four states. To have a detailed history of the required steps, we allow one transition between each consecutive state; thus, between states S and S', Roomba can move from the bedroom to the living room, or the attacker can switch the lock on or off, but both cannot occur simultaneously. One solution given by Alloy appears in Table 1. The order of the solutions provided by Alloy may differ between runs. Additionally, the solutions depend on the number of states; if a higher number of states is requested as a mandatory solution prerequisite, the solution will include a higher number of steps. Such solutions may have transitions that cancel each other, such as repeatedly moving the Roomba between the living room and the bedroom.

[11] We model time as a series of consecutive states.
[12] Alloy tutorial and info: http://alloy.mit.edu/alloy/tutorials/online/frame-FS-1.html

5.4 Performance Analysis

The experiments were performed on a 3.1 GHz Intel Core i7 processor with 16 GB RAM. For each round, we performe the experiment ten times and calculate the average CPU time. The number of clauses constructed for a particular configuration drives the memory consumption of the SAT solver SAT4J for that configuration. The results of our experiments are summarized in Table 2 and Figure 6. In the evaluated scenario, we model a single attacker. We assume the devices are on the same network and in two different physical locations.

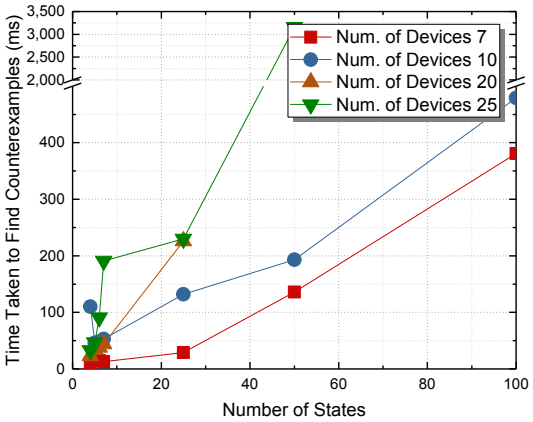
The SAT solver, SAT4J, exhaustively enumerates all possible combinations of devices and device states, and constructs clauses for these conditions. As expected, the number of clauses increases exponentially with the number of devices. This is supported by CPU and memory overhead measurements.

For smaller networks, although the number of clauses grew steadily, the impact on the CPU time was minimal. However, as the number of devices increases, the CPU time increases exponentially, as illustrated in Table 2. For the configuration with the largest number of clauses, with 25 devices and 50 states, the CPU time is a reasonable 37 seconds. Every time a new node is introduced, this step is repeated. When the number of states was increased to 100, the memory was not sufficient. This upper bound depends on the device on which the experiments are performed. We note that our method is applicable for networks with smaller number of nodes and states, and does not scale to large networks. Figure 6b shows how the state space explodes when the number of devices reaches 25 and the number of states is increased linearly.

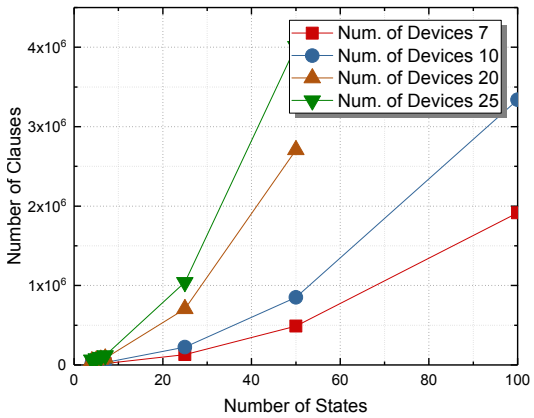
While the Alloy specification languages offers rich expressiveness, the theorem-proving strategy of Alloy restricts its applicability to scopes of limited size. In the future, we plan to investigate alternative options, including Prolog, that offer higher scalability. Initial results show that Prolog's backtracking approach may be superior in terms of performance but more analysis is needed.

6 DISCUSSION

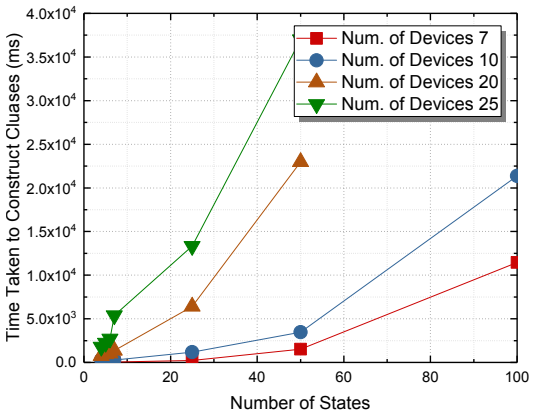
In this paper, we propose a way of modeling cyber and physical interactions between IoT devices of a given network. Our approach considers several noteworthy properties. First, by intertwining the real-time device identification into the system, our approach can identify and verify possible device interactions against user-defined



(a) Calculating the amount of time taken to find a counterexample to the `noSecurityDegradation` assertion.



(b) Calculating the number of clauses generated for a specific number of device and states.



(c) Calculating the amount of time taken to construct a set of clauses for a given number of devices and number of states.

Figure 6: Analyzing the CPU and memory consumption of Alloy for our models when the number of devices and states are varied.

policies in real time. Additionally, our model can be used to perform a differential assessment of two IoT networks. Specifically, we are interested in identifying how the addition (or removal) of an IoT device into an existing network impacts the interactions between devices and, consequently, the set of possible attack paths. Such information can be useful to determine if a particular device serves as a bridge between isolated IoT sub-networks, enabling new attack paths.

Our approach has a few limitations. Currently, our approach assumes that it is possible to identify all IoT devices and infer their location by passively sniffing wireless communications. This is a fair assumption since IoT devices are restricted in power and computing capabilities and rely on hubs or clouds to perform most of their tasks. Device fingerprinting is, however, non-trivial. For completeness, it must be able to differentiate between device versions (since new device version could introduce hardware changes) and instances (especially for mobile IoT devices).

In attempts to support various threat models, we take a "pessimistic" approach. Consequently, our model allows users to consider attackers of various capabilities, from attackers that can only influence input to attackers that can fully control the device. In doing so, however, our model fails to consider the limitations in behavior (input/output functions) imposed by the software operating on the devices. In other words, we assume that all possible combinations of input/output channels are possible. The software running on the IoT devices, however, may only enable certain input/output channel pairs. For example, a smart smoke detector equipped with a CO_2 sensor and a motion sensor, will sound the alarm if smoke is detected but not if motion is detected. Some of the attack paths identified by the model can, therefore, be considered false positives since they are not feasible without modification of the devices' software. To mitigate this issue, in the future, we wish to explore two options. One option is to couple our approach with a classification system that filters unrealistic paths. Another option is to expand the

Table 2: CPU time analysis raw data. We made use of the SAT4J solver built into the Alloy4 framework. We set the following parameters for the other variables - `<Attacker=1,Network=1,Location=2,Space=4,Access=4,PowerState=2>`, and checked for the property `noSecurityDegradation`. Based on our evaluation, the complexity increases in a non-linear manner, and this approach will not scale for high numbers of devices and states.

Number of Devices	Number of States	Clauses	Construction Time (ms)	Counterexample Time (ms)
6	5	11667	52	14
7	5	13915	49	17
7	25	130715	222	29
10	5	21595	206	46
10	25	224975	1181	132
20	5	57335	950	34
20	25	705315	6399	226
25	5	81055	2251	47
25	25	1041335	13338	230
25	50	4013560	37010	3157

model so that devices are defined by a set of input/output functions as defined by their software. The approach presented in this paper also shares the innate limitations of all SAT solvers. As discussed in Section 5.4, the processing time increases non-linearly with the number of devices and states in the Alloy implementation. In the future, we wish to explore alternative options that provide better scalability, making the approach practical even for industrial IoT networks.

7 CONCLUSION

Assessing and modeling the security of IoT systems is challenging because of their cyber-physical nature and heterogeneity. In this paper, we propose a way to model both cyber and physical interactions among IoT devices and consider that physical interactions are "channels" through which malicious inputs can be passed, denial of service triggered and other attacks can occur. These physical interactions include and go beyond the ones expected by the functional requirements of the system (a speaker communicating to Amazon Echo is not a functional specification, but is a channel). To validate the viability of our ideas, we implemented a proof-of-concept model using model checking tools (Alloy). Our model was able to cover attacks that would be undetected if model interactions were limited to cyber connections or the physical interactions already foreseen by the functional design of the system. During our evaluation, we identify that due to the size of the search space, SAT solvers explore all possible combinations and although extremely powerful, it hinders scalability of our model to only small IoT settings such as smart-home environments.

REFERENCES

[1] Ioannis Andrea, Chrysostomos Chrysostomou, and George Hadjichristofi. 2015. Internet of things: Security vulnerabilities and challenges. In *Computers and Communication (ISCC), 2015 IEEE Symposium on*. IEEE, 180–187. https://doi.org/10.1109/ISCC.2015.7405513

[2] Juan Carlos Augusto and Miguel J. Hornos. 2013. Software simulation and verification to increase the reliability of Intelligent Environments. *Advances in Engineering Software* 58 (2013), 18 – 34. https://doi.org/10.1016/j.advengsoft.2012.12.004

[3] Fulvio Corno and Muhammad Sanaullah. 2014. Modeling and formal verification of smart environments. *Security and Communication Networks* 7, 10 (2014), 1582–1598. https://doi.org/10.1002/sec.794

[4] A. Coronato and G. D. Pietro. 2010. Formal Design of Ambient Intelligence Applications. *Computer* 43, 12 (Dec 2010), 60–68. https://doi.org/10.1109/MC.2010.335

[5] D. Geneiatakis, I. Kounelis, R. Neisse, I. Nai-Fovino, G. Steri, and G. Baldini. 2017. Security and privacy issues for an IoT based smart home. In *40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 1292–1297. https://doi.org/10.23919/MIPRO.2017.7973622

[6] D'Arcy Gue. 2017. IoT Devices Top a Long List of 2017 Security Threats. http://www.medsphere.com/blog/iot-devices-top-a-long-list-of-2017-security-threats. *Medsphere* (2017).

[7] T. L. Guilly, J. H. Smedegård, T. Pedersen, and A. Skou. 2015. To Do and Not to Do: Constrained Scenarios for Safe Smart House. In *International Conference on Intelligent Environments*. 17–24. https://doi.org/10.1109/IE.2015.11

[8] Norm Hardy. 1988. The Confused Deputy:(or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review* 22, 4 (1988), 36–38.

[9] Daniel Jackson. 2002. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 2 (2002), 256–290.

[10] M. Kovatsch, Y. N. Hassan, and S. Mayer. 2015. Practical semantics for the Internet of Things: Physical states, device mashups, and open questions. In *5th International Conference on the Internet of Things (IOT)*. 54–61. https://doi.org/10.1109/IOT.2015.7356548

[11] Ulf Lindqvist and Peter G Neumann. 2017. The future of the Internet of Things. *Commun. ACM* 60, 2 (2017), 26–30.

[12] Richard P Lippmann, Kyle W Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Michael Artz, and Robert Cunningham. 2005. Evaluating and strengthening enterprise network security using attack graphs. *Lexington, Massachusetts October* (2005).

[13] O. Mavropoulos, H. Mouratidis, A. Fish, and E. Panaousis. 2017. ASTo: A tool for security analysis of IoT systems. In *IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*. 395–400. https://doi.org/10.1109/SERA.2017.7965757

[14] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. R. Sadeghi, and S. Tarkoma. 2017. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2177–2184. https://doi.org/10.1109/ICDCS.2017.283

[15] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman. 2016. IoTSAT: A formal framework for security analysis of the internet of things (IoT). In *IEEE Conference on Communications and Network Security (CNS)*. 180–188. https://doi.org/10.1109/CNS.2016.7860484

[16] Steven Noel and Sushil Jajodia. 2008. Optimal IDS Sensor Placement and Alert Prioritization Using Attack Graphs. *Journal of Network and Systems Management* 16, 3 (01 Sep 2008), 259–275. https://doi.org/10.1007/s10922-008-9109-x

[17] S. R. Oh and Y. G. Kim. 2017. Security Requirements Analysis for the IoT. In *2017 International Conference on Platform Technology and Service (PlatCon)*. 1–6. https://doi.org/10.1109/PlatCon.2017.7883727

[18] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. 2006. A Scalable Approach to Attack Graph Generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*. ACM, New York, NY, USA, 336–345. https://doi.org/10.1145/1180405.1180446

[19] Antonino Rullo, Daniele Midi, Edoardo Serra, and Elisa Bertino. 2017. A Game of Things: Strategic Allocation of Security Resources for IoT. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*

(IoTDI '17). ACM, New York, NY, USA, 185–190. https://doi.org/10.1145/3054977.3055001

[20] Bruce Schneier. 1999. Attack trees. *Dr. Dobb's journal* 24, 12 (1999), 21–29.

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

[22] A. Tekeoglu and A. Tosun. 2016. A Testbed for Security and Privacy Analysis of IoT Devices. In *IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. 343–348. https://doi.org/10.1109/MASS.2016.051

[23] Ashish Tiwari. 2012. HybridSAL relational abstracter. In *Computer Aided Verification*. Springer, 725–731.

[24] Lingyu Wang, Anyi Liu, and Sushil Jajodia. 2006. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications* 29, 15 (2006), 2917 – 2933. https://doi.org/10.1016/j.comcom.2006.04.001 Computer Communications.

[25] Rolf H Weber. 2010. Internet of Things–New security and privacy challenges. *Computer law & security review* 26, 1 (2010), 23–30.