Modeling Internet-of-Things (IoT) Behavior for Enforcing Security and Privacy Policies

Anubhav Gupta[1], Daniel Campos[2], Parth Ganeriwala[2], Siddhartha Bhattacharyya[2], TJ OConnor[2], and Adolf Dcosta[2]

[1] University of British Columbia, British Columbia, V1V 3C8, CA,
[2] Florida Institute of Technology, Melbourne, Florida, 32901, USA

Abstract. The availability and usage of the Internet of Things (IoT) have grown significantly over the last decade. This growth in ubiquitous computing has enabled continuous observation, decision-making, and execution of actions to improve the livelihood of millions. However, IoT also has an increased cybersecurity risk by making the user vulnerable to attacks from the outside or disclosing information without the user's knowledge. As a result, it becomes essential to understand the behavior of IoT devices and then model the behavior at a higher level of abstraction that enables automated reasoning, reducing the gap between policy requirements and its analysis. Towards this end, we have developed our proposed framework, which combines policy requirements guided structured experimentation performed on the IoT devices, followed by modeling to store the knowledge for analysis. We leverage the experimental findings to formulate the relationship of IoT devices in the model. The creation of this model enables automated analysis to identify if there is a potential violation of security or privacy. We demonstrate our approach by investigating the behavior of IoT devices and performing analysis to check if the IoT devices execute risky cybersecurity behavior.

Keywords: Ontology, IoT, Cybersecurity, Privacy, Policy Requirements

1 Introduction

With the innovations in communication technologies, hardware designs, software systems, and artificial intelligence, Internet of Things (IoT) has enabled applications that benefit day-to-day activities. As a result, IoT has become an essential part of our daily lives. It is further envisioned that the future of the internet will consist of heterogeneously connected devices that will further extend the borders of the world with physical entities and virtual components [35]. Although, IoT has enhanced our capabilities, it has also increased our vulnerability to a wide variety of cybersecurity attacks.

Some of the primary reasons for IoT devices to be more vulnerable are that these devices operate on wireless networks, thus increasing the ease with which confidential information is available to attackers by eavesdropping. IoT devices mostly communicate information without an operator monitoring or observing the data being sent. Based on the nature of IoT devices, complex security methodologies are hard to implement [29].

Advancements in IoT related technologies have increased the availability and affordability of IoT devices. These devices are being used for home automation or fitness or conducting daily activities, without the knowledge of the vulnerabilities or awareness of privacy and security issues. As a result, it is critical to identify the vulnerabilities posed by IoT. In regards to the identification of the private data accessed by IoT, the types of data accessed, the types of unauthorized data accesses possible or the alarming behavior of IoT in collecting data, there is either lack of awareness in the vulnerable behavior executed by these IoT devices or vendors collect information without the knowledge of the consumers. This becomes critical, as the IoT devices are used for defense or safety-critical operations [18].

One of the major challenges faced in this area of research is in the identification and extraction of relevant information for IoT devices and their composition, pertaining to cybersecurity, which allows the representation of well-defined concepts that are realistic and facilitate computer-based analysis. Towards this end, we discuss our approach which investigated the creation of a framework that elaborates on the process of abstracting important information to perform automated analysis.

In this research paper, related work is described in section 2, the research background is discussed in section 3, and our proposed methodology in section 4. Then in section 5 the policy requirements that guide the experimentation are discussed followed by modeling. In section 6 the experimental framework is explained. Section 7 discusses the actual model, followed by the automated analysis performed in section 8. Finally the results are discussed in section 9 with conclusion in section 10.

2 Related Work

With the exponential increase in the use of domain knowledge applications that require immense communication of information between systems, grasping the structural complexity of these systems along with their semantic relationships between the data and devices require models. To represent these models, ontologies are developed to represent data as concepts and formally specify the relationships between them. Ontologies enable the reuse of an agreed-upon collection of domain knowledge [24]. Li et al.[2] established the social attributes of things, evaluated the functionality of relations, which is one of the essential social attributes in IoT, and utilized super network architecture to portray the complex relationships among physical objects. They incorporated an ontology-based method to represent the relations of objects based on these relations and relation architecture. Ye et al.[39] presented a top-level ontology model which was used to represent the domain knowledge. They introduced concepts for common semantics in information at different levels to support the communication, re-use and sharing of ontologies between systems. They were restricted by the lack of a reasoner to implement their rules generated as part of their work. Rahman et al. [28] present a lightweight ontology with dynamic semantics that tackles semantic interoperability difficulties in different IoT implementations. The ontology significantly decreases query response time and computational resource requirement when compared to existing heavy-weight and complicated ontologies, while only including the most often used concepts in IoT. Their proposed ontology is an extension of IoT-Lite [21] and an abstraction of SSN ontology which is suitable for real-time IoT environment. The proposed methodology automatically concatenates a new node under a certain cluster based on the similarity index, making it suitable for a dynamic environment. However, they have not designed a light-weight generic ontology with dynamic semantic which has been left as future work.

The Internet of Things (IoT) is an integral part of daily activities. IoT systems are composed of smart devices that interact and transmit data without the need for human involvement. Due to the sheer development and autonomous nature of IoT systems, these devices are exposed and vulnerable to significant risks. Thus behavior capturing, and verification procedures are required to show the trust-level of these devices [1]. Wang et al. [37] brought the primary focus of existing research to be on device modeling without access and utilization of information being considered. They discussed the development of a robust description ontology for knowledge representation in the IoT domain and explored how it can be applied to facilitate tasks such as service discovery, testing, and dynamic composition. However, they failed to bring in the information security service into their ontology. Hachem et al. [16] have focused on addressing challenges with the scalable, heterogeneous nature of IoT devices and have integrated these concepts to formulate a set of ontologies that give a robust description of the devices and as well as, their functional relationships. They have also modeled the domain of physics, which is the foundation of the IoT, since it allows for the approximation and estimation of functionality often represented by things. This approach however needed a modeling language that was not too detailed to implement their ontology for evaluation. SIoT [9] was developed as an approach to model the social relationships among the IoT devices and to give access to humans to discover, select and use objects with the services in IoT bringing forward human trust analysis. De et al. [12] presented a semantic modeling approach for different components in an IoT framework. Their model could be integrated into the IoT framework by using automated association mechanisms with physical entities and thus the data could be discovered using semantic search and reasoning mechanisms. Koorapati et al. [20] proposed a method to draw actionable insights through operational analytics. While most IoT research focuses on data from IoT services and sensors, they focus on the variety of metadata that comprises the end-to-end IoT ecosystem. This study shows how to represent an end-to-end IoT ecosystem utilizing a semantic-based approach such as ontologies.

The SensorData Ontology developed in [38] is built based on Sensor Model Language (SensorML) specifications. The primary focus of SensorML is to provide a robust and semantically-tied means of defining processes and processing components. Although the language provides an appropriate modeling approach, it is too detailed for specific use-case ontologies to be modeled and evaluated. Sommestad et al. [31] have developed a cyber security modeling language(CySeMoL) for security threat analysis to assess the probability of attacks on modeled systems. It does not assess confidentiality attacks as confidentiality is of lesser importance in industrial-control systems than IoT. Thus, there is a need to introduce a model based formal language for modeling IoT systems for information security and privacy analysis.

3 Background

3.1 IoT vulnerabilities

With the widespread use of IoT devices in both homeland and deployed settings, there is a range of different IoT security threats that are relevant. For example, Fitbit fitness trackers can provide geolocation data to foes [18], and wifi cameras can send unauthorized video from secure areas [11]. The state of IoT is rife with vulnerabilities and poorly designed devices and ecosystems [25][19][13][14][36][22][26]. As such, there is a need for military, commercial, and civilians to be aware of these threats and ensure that everyone is equipped with the necessary knowledge, tools, or features to combat such threats. To this end, there is a need to conduct research to understand the threat that IoT devices pose and to identify cyber approaches across the full spectrum of operations that can mitigate these threats. This challenge demands further study to understand the interaction among IoT devices and generate cybersecurity defenses, both tactical and strategic, to ensure the secure and effective use of IoT devices.

Given the nature of vulnerabilities, Zero trust philosophy works well the IoT environment. Significant research efforts have been conducted that focus on perimeter protection based on the philosophy of trusting insiders and distrusting outsiders of a network in developing solutions for cybersecurity such as firewalls. The philosophy of zero trust focuses on "Never Trust Always Verify". Recent research in cybersecurity emphasized the need to design policies to detect and prevent insider threats in cyber infrastructure. Schultz [30] defines insider attackers as individuals who have been assigned privileges and the authority to execute their responsibilities. Still, they use that authority to damage the system or steal sensitive data. So, it is critical to design the system or software with zero trust philosophy. The behavior of each device can then be checked to identify the needed policies.

3.2 IoT Vulnerability Lifecycle

Rapid firmware development without any concern for the software development lifecycle is often the resulting cause of vulnerabilities in IoT. With the relatively low cost of IoT devices, vendors prioritize rapidly delivering a product to market and selling subscription-based services over software maintenance and developments [33]. A 2019 study by Parker Thompson examined over three million binaries from IoT and uncovered that vendors are failing to implement basic hardening features, including decades-old best practices [33]. Our experiences echo this concern. In the course of our experiments, we identified and reported six Common Vulnerabilities and Exposures (CVEs) for the Geeni and NightOwl branded WiFi camera doorbells [7][6][5][4][8][3] In each case, we informed the vendors of vulnerability, the root cause behind the vulnerability, and offered countermeasures to protect against future attacks. Geeni mitigated a single vulnerability with a firmware update [7] and discontinued the product line associated with the three other vulnerabilities [6][5][4]. NightOwl did not respond to any of our reports about the vulnerabilities in their product lines [8][3]. Despite five of the vulnerabilities currently being unpatched, no warning appears in either companion application to inform the user that they are using a vulnerable or discontinued product line.

3.3 Ontology

Ontologies can model data as concepts and formally identify the associations among them. We can leverage ontologies to create well-defined concepts that help researchers exchange information about a particular domain. In addition, ontologies provide reusability for an agreed-upon set of domain knowledge [24]. Several languages are used to develop ontologies. For example, Resource Description Framework (RDF), Ontology Inference Layer (OIL), Unified Modeling Language (UML), and Web Ontology Language (OWL) [17][10]. Protégé is a graphical environment that can be used to create an ontology. Protégé also supports the use of the OWL language [23]. Cybersecurity ontologies can be used to detect software vulnerabilities and their relationships. Undercoffer et al. [27] have been working on an Intrusion Detection System (IDS) ontology. The IDS ontology analyzed more than 4000 classes of computer attacks and how these attacks can be performed. The IDS ontology was then extended by Finin et al. [32] to include information integration and to write general rules, and it became the Unified Cybersecurity Ontology (UCO).

4 Proposed Methodology

Research indicates that modeling behavior of IoT is important in order to formulate cybersecurity policies. In this research effort, we come up with a proposed methodological framework (Figure 1) that helps to model the behavior of IoT for generation of cybersecurity policies. We begin by generating ”Policy Requirements” for the IoT devices that can help to increase user trust and privacy. For instance, a user might be interested in knowing if the IoT device used by him/her interacts with a cloud based server which is located in a country outside his/her country of residence. After the requirements have been generated, we go ahead and "Select" IoT devices that will be used for experimentation. We then develop the experimental framework and perform experiments on the selected IoT devices.

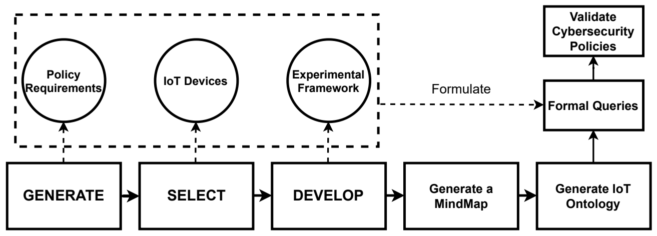


Fig. 1. Proposed Methodological Framework

The experiments performed on the selected IoT devices aids the process of generating requirements for the Mind Map. A Mind Map is a diagram which is used to visually organize information. The information gathered from the experiment is transferred to the Mind Map which, helps to visually organize the information and makes it much more easier to understand and comprehend. It lists all the entities along with their relationships and the associations between different entities. Using the entities and relationships, we come up with a formal IoT ontology model which helps to run automated reasoning and draw inferences. These experiments along with policy requirements and device selection help to formulate queries, that are executed on the ontology to infer knowledge from the ontological model. The outcome of the query reasoning helps to validate the policies and detect any kind of violations.

Our contribution in this research has been the formulation of an innovative policy guided experimental framework to capture IoT interactions. We have also developed a method to label network data to classify IoT security vulnerabilities in identifying the behavior of the IoT devices. Furthermore, we demonstrated the modeling of the behavior of IoT devices at a higher level of abstraction that was obtained from experimental outcomes which, were pertinent to security and privacy policies. .

5 Generation of Policy Requirements that Guide the Model Development

One of the main focus of this research effort is to aid the process of understanding the need for policies that help in increasing IoT trust and privacy. The aim is to develop a knowledge base that models the behavior of IoT devices for automated analysis, which is guided by some of the already existing and identified vulnerabilities related to security and privacy. For instance, in 2018, the US Military announced to reexamine its security policies after fitness tracker data was shared on social media that revealed bases and patrol routes [18]. Many such similar data privacy breaches have raised serious concerns regarding security policies with respect to IoT. Through this research effort, we aim to come up with a policies guided approach to cybersecurity, that can help identifying the relevant elements and relations among them that need to be modeled, so that possible violations of the policies can be captured if present in the model.

In-order to do so, we begin with the generation of policy requirements which have been categorised into four different categories:

– **Interaction**: Does the system interact with external systems (Systems located outside a defined geographical location)?
– **Server Ownership**: Who owns the server with which the system is interacting continuously?
– **Frequency**: How often does the communication/packet sharing happen?
– **Size and Type (Secured or Unsecured) of Data**: What is the size and type of data that is being shared by the system?

Policies pertaining to "Interaction" primarily focus on the interactions between the system and the outside world. They help to identify whether the system is interacting with any external servers and if the servers are trustworthy. For instance a user might be interested in knowing if the IoT device used by him interacts with a server which is located outside his/her country of residence. Similarly, network packet monitoring might be of importance and knowing the periodicity of communication can reveal some important insights about the system. Moreover, knowledge about the "Type and Size of Data" being shared by the device also helps to draw some important inferences relative to security of the system. Knowing if the shared data is secured or unsecured is important, since it will help to detect and prevent security vulnerabilities within the system. The knowledge about "Server Ownership" is also useful as the user might be interested in knowing whether the device interacts with a server that is owned by the manufacturing company of the device or does the company outsource it to a third-party cloud based company. All these policy requirements not only increase user trust and privacy but, also help to identify any type of violations or security breaches.

6 Experiments

Through the experimentation process, this study aims to come up with an abstract representation of the behavior of an IoT device that, can be represented as an ontology, so that we can build a knowledge base that can be used to perform automated reasoning to guide the generation of secure policies. The *IoT Lab* at *Florida Tech* was setup and configured to perform experiments on fifty six different types of IoT devices. As part of an ongoing experiment to automate device flow labels in *Florida Tech's IoT Lab*, IoT traffic was captured over a period of seven days on fifty six devices. All the communication between the devices present inside the lab and the outside world (network traffic entering and exiting the lab) was saved as packet capture files on a monitoring server while vendor APIs were periodically queried for event labels. For this study, we have particularly chosen Smart Doorbell IoT device because we believe that it will allow us to model an OWL representation which can be further generalized to represent any type of multimedia IoT device.

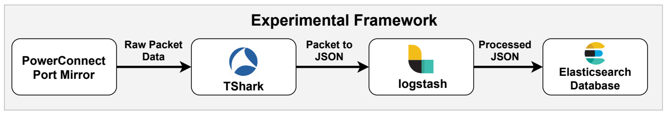


Fig. 2. Packets are ingested and processed into a query-able format for analysis

Figure 2 above represents the experimental framework. To start with, a Port Mirror is used to generate packet data in the raw format. Since the experiments were performed inside the University, Port Mirror helped in mirroring the packets and allowed for a transparent packet capture thereby emulating a home environment. This approach allowed to capture network packets from a setup that used the same components of the network in a passive and transparent manner. For interactive analysis, captured data is fed to Wireshark's command line processor, TShark (a network protocol analyzer), which converts the captured packet data into JSON objects. These JSON objects are then processed into Elasticsearch's logstash (server side data processing pipeline), which buffers and post processes the ingested data. This results in network packet data which are in the form of processed JSON objects that can be stored inside a database and queried for analysis. Finally, this information is stored onto Elasticsearch database so that it can be analyzed in Kibana (a query language and data visualization dashboard for ElasticSearch), which allows us to perform aggregations and visualizations on each device separately and independently.

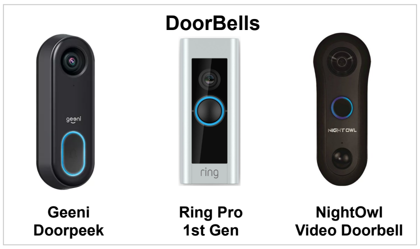


Fig. 3. Doorbells used for Experimentation

For this particular study, three different doorbells (Figure 3) were selected for analysis based on *"availability"*, *"observed maturity"*, and *"vendor popularity"*. The devices were obtained by purchasing through popular online retailers and physical stores. To represent major vendors, a Ring Video Doorbell Pro 1st generation was selected. We selected a Geeni Doorpeek 1080p Wireless Doorbell, and a NightOwl Video Doorbell as these were widely available. We term NightOwl's vendor as an *"immature"* vendor because it was realised during the experimentation that it employs industry wide bad-practises which could be due to lack of experience in the IoT security domain.

6.1 Hardware Analysis

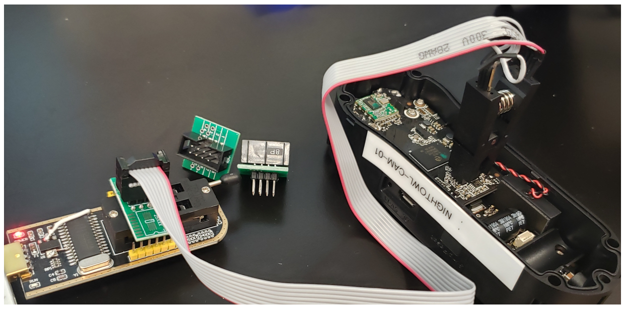


Fig. 4. A disassembled view of doorbell to identify high-level hardware components. Firmware can also be downloaded when applicable to identify network flows.

Vendors have a wide variety of hardware components that they can integrate in the final product. Generally, only major components are publicly identified, obscuring common dependencies or features that may exist across multiple devices. For this paper, we disassembled the three doorbells to identify common hardware patterns and components.

We discovered that the Night Owl and Geeni Doorbells shared the most components. Both utilized a "HiSilicon" based ARM processor and an "SPI-based" flash chip for storing the operating system software. Alongside these components, we discovered a "UART" serial console and a small battery source to keep the system online during intermittent power outages. All devices contained a removable camera module and usb-based WiFi module, as well as a USB port for powering the device without the use of a doorbell transformer.

- **Firmware Analysis**: During our analysis, we discovered a few calls from the HiSilicon based devices which we could not map to specific features. We previously identified that these devices utilized an ARM processor and an SPI flash chip for storing code. We hypothesized that an offline SPI programmer may be able to download the device firmware for offline analysis. Furthermore, if code signing or boot verification was not discovered, we could modify and upload the new firmware to perform dynamic analysis on the devices.

To perform the chip read and write, we utilized a CH341a SPI programmer as indicated in Figure 4. The programmer is available on several online retailers with a non-invasive clip to allow chip interaction without removing components. Firmware was downloaded and extracted using popular tools like Binwalk and Binary Ninja to perform extraction and analysis. Additionally, both devices contained telnet daemons and hard-coded admin credentials for remote management, which we overwrote to allow local access for dynamic analysis.

To simplify explanations in the following sections, we define the following categorizations of requests:

- **Demographically Internal** - Network requests which stay within a country's borders.
- **Demographically External** - Requests occurring outside a country's borders. These requests have legal implications on how data is handled.
- **First Party** - Any server or endpoint which stays under a vendor's root domain. For example, Ring subdomains es.ring.com and lpd.ring.com are considered first party.
- **Third Party** - Any server or endpoint which exists under another company's root domain. Tuya Smart, a popular IoT platform, hosts its endpoints under its own domain.

This approach is cost-effective, but can introduce unintended weaknesses as many developers either re-use existing code solutions or utilize a turnkey provider to develop the code base for them. During our analysis, we discovered that all of our HiSilicon based devices stored ARM binaries in an embedded SPI-based flash chip which varied from 4 - 8 MBs in size, alongside an open and accessible UART console.

SPI-based flash chips use standard read and write interfaces which can be accessed using external hardware. We leverage this interface and a CH341a SPI programmer to download each device's firmware. During this process, we discovered that none of the HiSilicon based devices utilized any form of code signing or image verification, which allowed us to explore additional investigative techniques using static and dynamic analysis. Dynamic analysis was performed by modifying the device software directly. Using the SPI programmer, we flashed modified images which gave us unrestricted access to the devices by resetting the root password and enabling telnet.

Using this technique, we successfully identified previously unknown services on the NightOwl and Geeni doorbells which allowed us to map the spontaneous NightOwl UDP connections to P2P *brokers* and identifying the types of information sent by Geeni requests.

6.2 Generalized Observations

As indicated in Table 1, we discovered variations on APIs contacted, hosting providers used, the use of encryption, and the demographic presence of all three devices. Most notably, Ring appeared to utilize services that were both demographically internal and first-party in nature. We contrasted this with the Geeni doorbell, where services appeared to be completely third-party in nature, but demographically internal. Night Owl used a mix of first and third party services. We observed similar patterns in the use of hosting providers when first party sources were used. All vendors used Amazon Web Services almost exclusively, but Night Owl included an additional hosting provider for their first party sources.

Table 1. Experimental Observations

Device	**First/Third Party**	**Host/Vendor**	**Demographic Presence**
Ring Video Doorbell Pro	First	Amazon AWS	Internal
Geeni Doorpeek	Third	Tuya, AWS	Internal
Nightowl Doorbell	Mixed	First Party - Amazon AWS, Liquid Web	
		Third Party - Unknown / Private	Mixed

Ring transmitted the largest amount of data, with Night Owl following closely. These transmissions occurred primarily on event boundaries and were destined for identified CDN endpoints which mirrors audio and video style uploads. This is contrasted with the Geeni Doorbell, which sent the least amount of data. The Geeni doorbell only generated small amounts of data which could coincide with single frame uploads. We also observed several time-bounded flows which repeatedly occurred when dealing with update and API endpoints.

6.3 Generalized IoT Environments

For the following sections, specific environment traits are compared with the typical IoT model which separates CDNs, API, and Core servers from users and end devices. For all data tables, DNS and NTP traffic was filtered. While both are necessary for device function, neither significantly contribute to any of our findings.

Ring Doorbell Pro The Ring Doorbell recorded a total of 341 event labels over the course of the experiment. Of these events, 33 were generated by physically ringing the doorbell, and 308 were generated by triggering the device motion sensor. Along these events, we identified a total of 1,720 flows. 648 flows were performed over UDP connections to what appeared to be CDN server pools. 4 TCP connections were made to lpd.ring.com and were long-lived in nature, mirroring Core Server / idle connection functionality for on-demand events. 419 TCP connections were made to fw.ring.com, performing what we believe were firmware update checks. LPD and FW connections transferred less than 30MB of data in a 7-day period, emphasizing the possibility of API-exclusive data.

Es.ring.com was resolved using the DNS alias event-sink-gw.ring.com. We observed the creation of 250 TCP connections which occurred around event boundaries. Only one connection stayed open for more than 3 seconds, with a large majority closing within 0-1 seconds. Most data was transmitted to unresolved addresses over 399 TCP and 648 UDP connections for 1GB and 5GB of data respectively. For both categories of data, we observed connection times typically ranging from 10-60 seconds mirroring typical media upload behaviors.

Table 2. Ring Doorbell Experimental Observations

DNS	**Encrypted**	**Load Balanced**	**TCP / UDP Flows**	**Total Data Sent / Received**	**Purpose**
lpd.ring.com	Y	Y	4 / 0	26.10 MB	API
es.ring.com	Y	Y	250 / 0	142.78 MB	API
fw.ring.com	Y	Y	419 / 0	4.66 MB	Updates
N/A	Y	Y	399 / 648	1.08 GB TCP / 5.20 GB UDP	CDN / Unk

206 IP addresses were returned as possible endpoints for es.ring.com, versus 352 possible endpoints for the CDN-styled endpoints. We contrast this with LPD, which only returned 19 possible endpoints. We hypothesize that a significantly larger pool of media-oriented endpoints exist to handle the larger amount of transmitted media. ES and LPD scale down accordingly which appears to correlate with the amount of handled data. Table 2 summarizes the behaviors observed during the experimentation process.

NightOwl Doorbell The NightOwl doorbell recorded 76 total events. Of these events, 29 were generated by physically pressing a button while 47 were generated by triggering the device motion sensor. While the doorbell press events were close to what we observed with the Ring doorbell, the motion events were drastically reduced. We hypothesize that NightOwl's decision to include a dedicated motion sensing hardware component caused this reduction. Many motion sensors were triggered by lab lights periodically cycling on and off. PIR based sensors were not activated by these events, mirroring the behavior observed in the NightOwl doorbell which corresponded with in-person movement events.

NightOwl spread connections over 4 first-party domains and 6 unresolved endpoints. Of the 4 first-party domains, both host.nightowl domains are hosted under Liquid Web LLC. Update and cloud-storage are hosted under Amazon Web Services. Of the 6 unresolved endpoints, 5 are used to facilitate P2P streaming of the camera's video feed. These endpoints are configured globally, with 2 endpoints located in China, 1 in the Netherlands, and the remainder in the United States. 1 unknown TCP endpoint hosts an insecure web server used to generate push notification events.

Update.nightowlsuper.com also hosted an insecure web server used to indicate when a software update was available. Over the course of the experiment, the NightOwl doorbell connected twice to this endpoint. The two host.nightowl.com endpoints maintained long-lived UDP connections which periodically transmitted data. The sent data was not deeply investigated, but very few bytes changed between packets indicating a lack of, or poorly implemented, encryption. The

Table 3. NightOwl Doorbell Experimental Observations

DNS	Encrypted	Load Balanced	TCP / UDP Flows	Total Data Sent / Received	Purpose
cloud-storage.nightowlconnect.com	Y	Y	124126 / 0	916.04 MB	API / CDN
host.nightowldvr07.com	N	N	0 / 3	69.11 MB	Unk
host.nightowldvr08.com	N	N	0 / 3	69.12 MB	Unk
update.nightowlsuper.com	N	Y	2 / 0	2.16 KB	Update
N/A	N	N	98 / 15	123.03 KB TCP / 7.84 KB UDP	PNS / P2P

only encrypted connections were made under cloud-storage.nightowlconnect.com, which was hosted on Amazon Web Services. Conveniently, this endpoint appears to connect exclusively over port 443/TCP, which is a common endpoint for web-based APIs. We observed 124,126 unique connections to this endpoint, with connection times ranging from 0 seconds to 2 days, with an average time of 3 seconds. Furthermore, almost 1GB of data is transferred during our experiment, indicating its use as a potential CDN on top of API data. Over all endpoints, cloud-storage.nightowlconnect.com appeared to be the only endpoint which utilized encryption. All others were either performed over insecure http or used a poor implementation on top of an unknown protocol. Table 3 summarizes the observations and lists the important data flows.

Geeni Doorpeek Doorbell The Geeni doorbell recorded a total of 232 events. Of these events, 38 corresponded to physical button presses, and 194 corresponded with motion events. This correlates with the Ring Doorbell reported events and matches our previous hypothesis that the Night Owl reduction is caused by the hardware sensor. Geeni also reports changing light conditions as motion, bolstering its numbers up to Ring's reported numbers which uses a similar technology. However, ring still reported 100 additional events that the Geeni doorbell failed to detect.

Table 4. Geeni Doorpeek Doorbell Experimental Observations

DNS	Encrypted	Load Balanced	TCP / UDP Flows	Data	Purpose
aws3nat.tuyaus.com	Y	Y	0 / 3	4.20 KB	P2P
aws4nat.tuyaus.com	Y	Y	0 / 3	3.17 KB	P2P
a2.tuyaus.com	Y	Y	633 / 0	3.40 MB	API
m2.tuyaus.com	Y	Y	39 / 0	6.59 MB	API
ty-us-storage30...	Y	Y	193 / 0	13.22 MB	CDN
N/A	Y	Y	38 / 0	1.94 MB	CDN

We did not detect any first-party domains for the Geeni doorbell. Instead, we observed the direct use of Tuya endpoints, AWS Storage gateways, and two direct AWS S3 endpoints over 903 TCP flows. Under Tuya, we discovered two endpoints, aws4nat and aws3nat, which are used to assist in P2P applications like video and audio streaming. These endpoints represent 6 UDP flows out of the 909 total. ty-us-storage30's 193 TCP connections and the 38 unknown connections resolve to Amazon Web Services S3 endpoints, which are used to upload and retrieve data in the cloud. We also observe that the sum of both services comes out to 231 unique connections, 1 event short of our reported event count. This bolsters our theory on its use as a CDN endpoint and also verifies that the device is only uploading to these endpoints on event boundaries. m2.tuyaus.com connections are exclusively MQTT which is traditionally used for long-lived idle connections for real-time commands on end devices. This mirrors Ring's use of lpd.ring.com for holding long-term connections to control devices. Finally, a2.tuyaus.com connects exclusively over port 443, leverage fast, on-demand connections for small bursts of data over 633 short-lived connections. This corresponds with firmware update checks, event data, or API requests. Table 4 summarizes the experimental observations.

7 Model

Formal representation capability for different kinds of knowledge is provided within an Ontology framework. A knowledge framework can be constructed and formalized for a specific ontology by leveraging an ontology language. OWL, endorsed by the W3C, is one of the most recent developments in standard ontology language. We use Protégé 5.5.0 [23], a free open-source ontology editor and knowledge base construction tool, to develop an IoT Framework and use automated reasoning to infer knowledge from already existing knowledge. It consists of different built-in elements such as Classes Tab, Object Properties Tab, Data Properties Tab, Individuals Tab, Rules Tab, etc. Protégé's most important feature is that it can be executed by a DL reasoner that provides for classification, consistency checking, and policy validation within the knowledge framework. In this work, we use Protégé to formally model and represent the requirements. We will graphically list the requirements through the Mind Map to generate an ontology as part of future work. We further go on to create instances, in order to build a knowledge base that can be reasoned using HermiT reasoner [15]. The inferences drawn from reasoning assist in validating the need for policies.

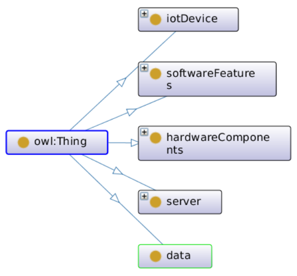


Fig. 5. OWL Thing Class, data properties, and object properties are omitted

To begin with, every model in Ontology contains a super class known as the *"Thing"* class which then flows down into other sub-classes. All other classes are sub-classes of the *"Thing"* class. The IoT device Ontology's Thing class has five child classes namely *"IoTDevice"*, *"server"*, *"softwareFeatures"* , *"data"* and *"hardwareComponents"* as seen in Figure 5. It is noteworthy that all five classes are disjoint with each other. The *"IoTDevice"* class is further decomposed into sub-classes as represented in Figure 6. The *"smartDoorBell"*, *"smartDoorLock"*

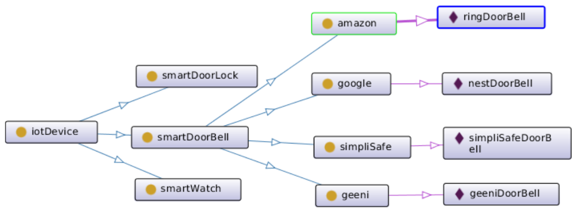


Fig. 6. OWL IoT Device Class, data properties and object properties are omitted

and *"smartWatch"* are child classes that inherit all the features and properties of the *"IoTDevice"* class. We mainly focus on the *"smartDoorBell"* and create instances with data properties for each of the vendors we experimented with. The data properties that have been generated during the experimentation process are used to formulate the knowledge base within the IoT Ontology. We can then perform automated reasoning on this knowledge base for consistency checking and validation. We try to list all important data properties that we believe will help in drawing insightful inferences. The *"IoTDevice"* class is disjoint with the *"server"* class indicating that a server cannot be an IoT device and vice-versa. The *"IoTDevice"* is associated with the *"server"* through "isConnectedTo" object property association indicating that each IoT device is connected to at-least one or more servers.

The *"server"* class is a representation of cloud based server that provides connectivity to IoT devices for data storage and other types of communication such as encryption-decryption, TLS handshaking, etc. It might be of relevance to know the location of the server with which the IoT device is interacting. Moreover, it is also important to know if the server is owned by the manufacturing company itself or it is being outsourced. Every instance of a server has certain data property attributes such as "encryptedConnection", "portNumber", "serverLocation", "thirdPartyorFirstParty", "demographicallyExternalorInternal" and "serverType" that define it's behavior.

Figure 7 represents the "softwareFeatures" class which is a subclass of *"Thing"* and is disjoint with the *"IoTDevice"* class. It represents various software features that are found in smart doorbells. The features have been categorised into three broad categories which are *"securityFeatures"*, *"internetFeatures"* and *"smart-*

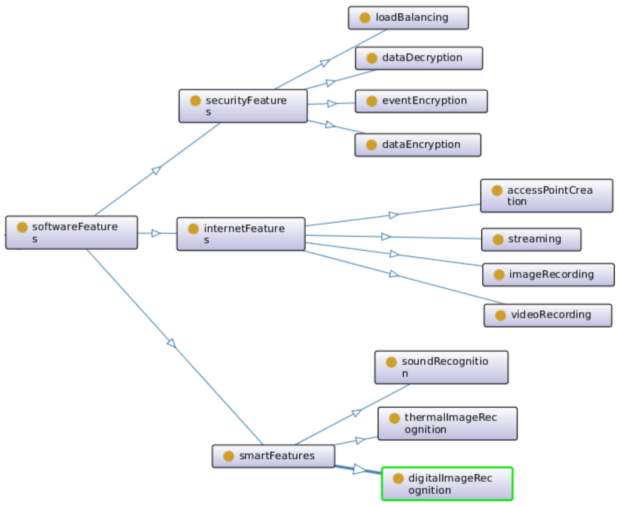


Fig. 7. OWL Software Features Class, data properties and object properties are omitted

Features". Each category is further sub-categorised and are indicative of the features that are present in the doorbells. Security features such as *"loadBalancing"*, *"eventEncryption"*, etc. make the device and communication secure. Similarly, the Internet Features such as *"streaming"*, *"imageRecognition"*, etc. indicate the general behavior of the doorbell. There are some bells that incorporate certain smart features into their functionality such as "soundRecognition", "thermalImageRecognition", etc. Every doorbell has a unique behavior and may contain one or more of such features. The data properties are formulated in a way that querying can indicate if a particular doorbell supports a feature or not. For instance, we can query to find out if the "Ring" doorbell performs *"eventEncryption"* or *"dataEncryption"*.

The *"hardwareComponents"* class shown in Figure 8 is disjoint with the *"IoTDevice"* class and represents the various hardware components that are present inside the doorbell. Majority of the doorbells have a "button", "microprocessor", "speaker", "camera" and "microphone". Some doorbells like the smart doorbells provide additional functionalities such as "heatSenor", "motionSensor", "thermalImageRecognitionSensor" and "networkModule". Depending on the components present inside a doorbell we have created instances of each and listed the data properties that help to create the knowledge base. Similarly, the *"Data"* class as shown in Figure 5 represents the data that is sent and received by the IoT device. It has attributes to store certain data properties such as the Type and Size of data which are essential to infer knowledge and come up with secure policies. The *"smartDoorBell"* class is connected to the *"Data"* class through an object property assertion *"sendsAndReceives"*.

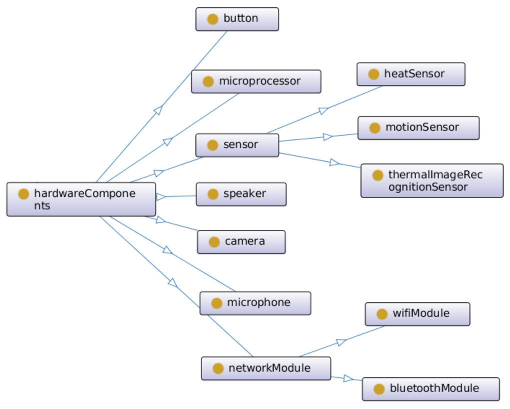


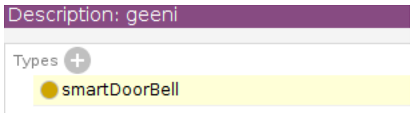
Fig. 8. OWL Hardware Components Class, data properties and object properties are omitted

In the next section, we discuss the outcomes and results of automated reasoning which helps to infer and list some interesting findings.

8 Automated Reasoning in Protege

Reasoning within ontology helps draw inferences and can be achieved using inbuilt reasoners. In its normal distribution, Protégé includes a variety of reasoners. HermiT is available in Protégé through a reasoner Plug-in thereby providing HermiT's special capabilities for checking ontology consistency. Reasoning helps infer new knowledge from the existing knowledge within an ontology. Classification is one of the most widely known usage of an automated reasoner. Classification is invoked as soon as we run the HermiT reasoner in Protégé. The classification process involves three steps. Firstly, it checks whether the structure of OWL ontology satisfies all axioms or not. If it does not satisfy all the axioms, it will classify the model to be inconsistent and return a warning. All inconsistencies have to be rectified inorder to move forward and perform reasoning. Secondly, it will check whether there exists a structure that satisfies all the axioms in the model and in which A has an instance, say y, where A is a class and y is an instance of any class. Lastly, for any two class names, say X and Y, that occur in the model, the reasoner tests whether each instance of X is also an instance of Y. The results of these tests are shown in the the Protégé OWL editor in the form of the inferred class hierarchy. For our model, Figure 9 graphically depicts the inferred class type for the instance geeni. Even though we do not explicitly specify it's type, the reasoner based on the inherited knowledge, defines it as a "smartDoorBell".

Consequently, Figure 10 graphically depicts the various property assertions and the inferred properties for the instance geeni. We see the *hasFeature softwareFeature1* assertion made for the instance geeni and the HermiT reasoner

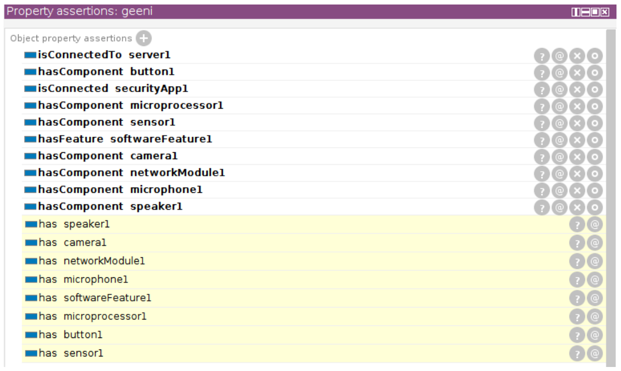


Description: geeni

Types

smartDoorBell

Fig. 9. Inferred Knowledge for geeni instance using HermiT Reasoner



Property assertions: geeni

Object property assertions

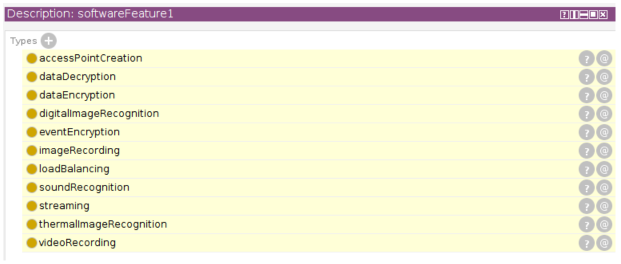
```
isConnectedTo server1
hasComponent button1
isConnected securityApp1
hasComponent microprocessor1
hasComponent sensor1
hasFeature softwareFeature1
hasComponent camera1
hasComponent networkModule1
hasComponent microphone1
hasComponent speaker1
has speaker1
has camera1
has networkModule1
has microphone1
has softwareFeature1
has microprocessor1
has button1
has sensor1
```

Fig. 10. Property Assertions and Inferred Property Knowledge for geeni instance using HermiT Reasoner

makes an inference *has softwareFeature1* automatically. Furthermore, Figure 11 details the inferred class type for the instance softwareFeature1. This inference, encompasses the IoT device feature properties, which enables query based analysis to identify if there is a potential violation of any security or privacy policy requirements. Such similar kind of inferences are drawn using the inbuilt reasoner and it also allows for inconsistency checking which are removed before we proceed with querying.

After classification using the HermiT reasoner, we perform automated reasoning on IoT Ontology framework using SPARQL [34] query to infer knowledge from the existing knowledge base. SPARQL [34] being a Resource Description Format (RDF) query language, allows to retrieve and manipulate information stored in ontology in RDF format.

SPARQL querying allows for more deductive powerful reasoning than OWL alone and provides similar strong formal guarantees when performing inferences. OWL allows for automated reasoning using different web rule languages such as SPARQL, SWRL, RuleML etc. We prefer SPARQL over SWRL and other querying languages because it is being used more extensively for automated reasoning and allows support with the latest version of Protégé. SPARQL allows querying of the entire knowledge base which is a set of *"subject-predicate-object"* triples. It specifies four different query variations for specific purposes: 1. **SELECT** query, 2. **CONSTRUCT** query, 3. **ASK** query and 4. **DESCRIBE** query. The "SELECT" query is used to extract raw values from a SPARQL endpoint, the "CONSTRUCT" query transforms the extracted information into


Description: softwareFeature1

Types

accessPointCreation
dataDecryption
dataEncryption
digitalImageRecognition
eventEncryption
imageRecording
loadBalancing
soundRecognition
streaming
thermalImageRecognition
videoRecording

Fig. 11. Inferred Knowledge for softwareFeature1 instance using HermiT Reasoner

valid RDF, the "ASK" query is a question query that returns true/false and the "DESCRIBE" query also helps to extract RDF graph from the SPARQL endpoint but it gives more freedom and flexibility to the maintainer. To query our model we mainly use the "SELECT" and the "ASK" query to infer knowledge from the knowledge base. The queries are executed against the knowledge base, which returns a list of tuples of ontology values that satisfy the query.

9 Results

In this section, we present the results of policy requirements that were validated through SPARQL reasoning and querying. As discussed in the previous sections, we began by listing and categorizing the policy requirements, which guided the experimental flow. The experiments discussed in Section 6 helped in providing the formal requirements for the ontology. We formally modeled these requirements after identifying the various components, attributes and their associations. After modeling, the automated reasoning helped to check for any inconsistencies and flaws within the framework which were rectified. As part of the last step, validation of the four main policies listed in Section 5 was achieved through querying. Table 5 lists the different categories of policy requirements along with

Table 5. Results of SPARQL Query

Requirements	**Query**	**Outcome**

1. **Interaction**	SELECT ?name ?server ?Demographics ?Location
	WHERE { ?name test:isConnectedTo ?server.
	?server test:demographicallyExternalorInternal ?Demographics .
	?server test:serverLocation ?Location}
		name, server, Demographics, Location
		
		1. **ring**, **server2**, **"Internal"**, **"USA"**
		2. **geeni**, **server1**, **"Both"**, **"USA, Belgium"**
		3. **nightOwl**, **server3**, **"External"**, **"China"**

2. **Server Ownership**	ASK {test:ring test:isConnectedTo test:server2 . test:server2 test:thirdPartyorFirstParty "First"^^xsd:string}	**True**

3. **Frequency**	SELECT ?name ?server ?FrequencyOfInteraction
	WHERE { ?name test:isConnectedTo ?server.
	?server test:frequencyofInteraction ?FrequencyOfInteraction}
		name, server, FrequencyOfInteraction
		
		1. **ring**, **server2**, **"continuous"**
		2. **geeni**, **server1**, **"periodic"**
		3. **nightOwl**, **server3**, **"discrete"**

4. **Size and Type (Secured or Unsecured) of Data**	SELECT ?name ?data ?SizeOfData ?TypeOfData ?Security
	WHERE { ?name test:sendsAndReceives ?data.
	?data test:totalSize ?SizeOfData .
	?data test:typeOfData ?TypeOfData .
	?data test:dataEncryption ?Security}
		name, data, SizeOfData, TypeOfData, Security
		
		1. **geeni**, **dataGeeni**, **"13.22mb"**, **"Multimedia"**, **"Unsecured"**
		2. **ring**, **dataRing**, **"26mb"**, **"JSON"**, **"Secured"**
		3. **nightOwl**, **dataNight**, **"51mb"**, **"Multimedia"**, **"Both"**

the SPARQL based queries and their outcomes. We first list the policy requirement for "Interaction" wherein the user might be interested in knowing if the system interacts with systems that are located outside a defined geographical region. The result of the query lists the name of the countries where the servers are located with which the system is interacting and sharing information. We then list the formal query for "Server Ownership" wherein the user might be interested in knowing the company that owns the server. The "ASK" query results in a Boolean value indicating that the ring doorbell is interacting with a server which is a first party server and is owned by the company itself. Next, the frequency policy requirement is translated into a formal representation. The result of the query lists the frequency of interaction between the doorbell and the server to which it is connected. For instance, the ring doorbell continuously interacts with the server to share information. Lastly, a user might be interested in knowing some properties of the data (size, type, and security) that is being shared by the device with the server. In total, we listed around fifty formal queries from different categories. The outcomes of these queries not only help to infer knowledge, identify potential security vulnerabilities but they can also help to validate the existing policies. The checking of violation of security, potential security vulnerabilities are important to identify because it'll help to formulate new policies in the future that are more secure and ensure user privacy. For instance, the users might want to restrict interactions with servers that are located outside a defined geographical region. This specifically becomes important when the devices are being used for special tasks or by special forces deployed by the national agencies. In order to ensure the same, a policy can be formulated that restricts any interaction with the outside world. We believe the proposed formalization and framework will help to bridge the knowledge gap or lack awareness that exists.

10 Conclusion

In this research effort, we hypothesized and formulated a framework that enables capturing relevant information regarding the behavior of IoT devices guided by requirements focusing on capturing security vulnerabilities. We then employed the captured experimental knowledge to develop an ontological representation at a higher level of abstraction to perform automated reasoning. Our effort also demonstrated the integration of experimental knowledge with formal modeling to enable reasoning at a level of abstraction closer to requirements. It includes modeling objects and associations for automated reasoning. The models allow future research efforts to focus on context-based mitigation of security vulnerabilities by including human-machine interactions.

11 Acknowledgments

This material is based upon work supported in whole or in part with funding from the Office of Naval Research (ONR) contract #N00014-20-1-2798. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ONR and/or any agency or entity of the United States Government.

References

1. Ali, J., Khalid, A.S., Yafi, E., Musa, S., Ahmed, W.: Towards a secure behavior modeling for iot networks using blockchain. CoRR abs/2001.01841 (2020), http://arxiv.org/abs/2001.01841
2. Ali, L., Xiaozhen, Y., Huansheng, N.: Thing relation modeling in the internet of things. IEEE Access 5 (2017), http://ieeexplore.ieee.org/document/8000306/
3. Anonymous, Anonymous: CVE-2020-2871. Available from MITRE, CVE-ID CVE-2020-2871. (Nov 24 2020)
4. Anonymous, Anonymous: CVE-2020-28998. Available from MITRE, CVE-ID CVE-2020-28998. (Nov 24 2020)
5. Anonymous, Anonymous: CVE-2020-28999. Available from MITRE, CVE-ID CVE-2020-28999. (Nov 24 2020)
6. Anonymous, Anonymous: CVE-2020-29000. Available from MITRE, CVE-ID CVE-2020-29000. (Nov 24 2020)
7. Anonymous, Anonymous: CVE-2020-29001. Available from MITRE, CVE-ID CVE-2020-29001. (Nov 24 2020)
8. Anonymous, Anonymous: CVE-2021-31793. Available from MITRE, CVE-ID CVE-2021-31793. (Mar 24 2021)
9. Atzori, L., Iera, A., Morabito, G.: SIoT: Giving a social structure to the internet of things 15(11), 1193–1195 (2011), http://ieeexplore.ieee.org/document/6042288/
10. Bechhofer, S.: OWL: Web Ontology Language, pp. 2008–2009. Springer US, Boston, MA (2009), https://doi.org/10.1007/978-0-387-39940-9_1073
11. Conversation, T.: Hackers can access your mobile and laptop cameras and record you – cover them up now (2020), https://tinyurl.com/mpffrxpe
12. De, S., Barnaghi, P., Bauer, M., Meissner, S.: Service modelling for the internet of things p. 7 (2011)
13. Dooley, E.: ADT Technician Pleads Guilty to Hacking Home Security Footage (Jan 2021), https://www.justice.gov/usao-ndtx/pr/adt-technician-pleads-guilty-hacking-home-security-footage
14. Fereidooni, H., Frassetto, T., Miettinen, M., Sadeghi, A.R., Conti, M.: Fitness trackers: fit for health but unfit for security and privacy. In: 2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE). pp. 19–24. IEEE (2017)
15. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An owl 2 reasoner. Journal of Automated Reasoning 53, 245–269 (2014)
16. Hachem, S., Teixeira, T., Issarny, V.: Ontologies for the internet of things. In: Proceedings of the 8th Middleware Doctoral Symposium on - MDS '11. pp. 1–6. ACM Press (2011), http://dl.acm.org/citation.cfm?doid=2093190.2093193
17. Horrocks, I., Patel-Schneider, P.F., Van Harmelen, F.: From shiq and rdf to owl: The making of a web ontology language. Web semantics: science, services and agents on the World Wide Web 1(1), 7–26 (2003)
18. Hsu, J.: Strava data heat maps expose military base locations around the world | WIRED (2018), https://www.wired.com/story/strava-heat-map-military-bases-fitness-trackers-privacy/

19. Janes, B., Crawford, H., OConnor, T.: Never ending story: Authentication and access control design flaws in shared iot devices. In: Security and Privacy Workshops (SPW). pp. 104–109. IEEE, San Francisco, CA (2020)
20. Koorapati, K., Pandu, R., Ramesh, P.K., Veeraswamy, S., Narasappa, U.: Towards a unified ontology for iot fabric with sddc. Journal of King Saud University - Computer and Information Sciences 34(8, Part B), 6077–6091 (2022), https://www.sciencedirect.com/science/article/pii/S1319157821001014
21. Maria Bermudez-Edo, P., Elsaleh, T., Taylor, K.: Iot-lite: A lightweight semantic model for the internet of things. International Conferences on Ubiquitous Intelligence & Computing p. 1 – 8 (2016), https://www.scopus.com/inward/record.uri?eid=2-s2.0-85097793434&partnerID=40&md5=5ca8613143917a43d95da97b303d0af7, cited by: 1
22. Mitev, R., Pazii, A., Miettinen, M., Enck, W., Sadeghi, A.R.: Leakypick: Iot audio spy detector. In: Annual Computer Security Applications Conference. pp. 694–705 (2020)
23. Musen, M.A.: The Protégé Project: A Look Back and a Look Forward. AI matters 1(4), 4–12 (2015)
24. Noy, N.F., McGuinness, D.L., et al.: Ontology development 101: A guide to creating your first ontology (2001)
25. OConnor, T., Enck, W., Reaves, B.: Blinded and confused: Uncovering systemic flaws in device telemetry for smart-home internet of things. In: ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec). pp. 140–150. ACM, Miami,FL (2019)
26. OConnor, T., Jesse, D., Camps, D.: Through the spyglass: Toward iot companion app man-in-the-middle attacks. In: Cyber Security Experimentation and Test (CSET). USENIX, Virtual Event (August 2021)
27. Pinkston, J., Undercoffer, J., Joshi, A., Finin, T.: A target-centric ontology for intrusion detection. In: In proceeding of the IJCAI-03 Workshop on Ontologies and Distributed Systems. Acapulco, August 9 th. Citeseer (2004)
28. Rahman, H., Hussain, M.I.: A light-weight dynamic ontology for internet of things using machine learning technique. ICT Express 7(3), 355–360 (2021), https://www.sciencedirect.com/science/article/pii/S2405959520304902
29. Roman, R., Zhou, J., Lopez, J.: On the features and challenges of security and privacy in distributed internet of things. Computer Networks 57(10), 2266–2279 (2013)
30. Schultz, E.: A framework for understanding and predicting insider attacks. Journal of Computers & Security 21(1), 526–531 (2002)
31. Sommestad, T., Ekstedt, M., Holm, H.: The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures 7(3), 363–373 (2013), http://ieeexplore.ieee.org/document/6378394/
32. Syed, Z., Padia, A., Finin, T., Mathews, L., Joshi, A.: Uco: A unified cybersecurity ontology. In: Workshops at the Thirtieth AAAI Conference on Artificial Intelligence (2016)
33. Thimpson, P.: Binary Hardening in IoT products (Aug 2019), https://cyber-itl.org/2019/08/26/iot-data-writeup.html
34. W3C: SPARQL: Query Language (2013), https://www.w3.org/TR/sparql11-query/
35. Wang, P., Valerdi, R., Zhou, S., Li, L.: Introduction: Advances in IoT research and applications. Information Systems Frontiers 17(2), 239–241 (2015)

36. Wang, Q., Datta, P., Yang, W., Liu, S., Bates, A., Gunter, C.A.: Charting the attack surface of trigger-action iot platforms. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1439–1453 (2019)
37. Wang, W., De, S., Toenjes, R., Reetz, E., Moessner, K.: A comprehensive ontology for knowledge representation in the internet of things. In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 1793–1798. IEEE (2012), http://ieeexplore.ieee.org/document/6296201/
38. Wei, W., Barnaghi, P.: Semantic annotation and reasoning for sensor data. In: Proceedings of the 4th European Conference on Smart Sensing and Context. p. 66–76. EuroSSC'09, Springer-Verlag, Berlin, Heidelberg (2009)
39. Ye, J., Stevenson, G., Dobson, S.: A top-level ontology for smart environments. Pervasive and Mobile Computing 7(3), 359–378 (2011), https://linkinghub.elsevier.com/retrieve/pii/S1574119211000277