SENSOR-BASED PROVENANCE FOR THE INTERNET OF THINGS AND AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

EBELECHUKWU NWAFOR

Department of Electrical Engineering and Computer Science

APPROVED:

Gedare Bloom, Ph.D.

Legand Burge, Ph.D.

Danda Rawat, Ph.D.

Charles Kim, Ph.D.

Gary L. Harris, Ph.D.
Dean of the Graduate School

to my

MOTHER and FATHER

whose endless love has inspired me to achieve greater heights

thanks for all the love and support, i am forever grateful

SENSOR-BASED PROVENANCE FOR THE INTERNET OF THINGS AND AUTOMOTIVE CYBER-PHYSICAL SYSTEMS

by

EBELECHUKWU NWAFOR, M.S.

DISSERTATION PROPOSAL

Presented to the Faculty of the Graduate School of

Howard University

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

Department of Electrical Engineering and Computer Science

HOWARD UNIVERSITY

FEB 2016

Acknowledgements

First and foremost, I would like to thank God for making this possible, without him i will not be where i am today. I would also like to thank my parents, Benneth and Chinwe for their constant encouragement, love and support.

I would like to thank my advisor Dr. Gedare Bloom for his guidance and encouragement. Also for believing in me and my research ideas. I would like to thank my lab partner Habbeeb Olufowobi and fellow graduate student Marlon Mejias for their constant constructive criticism and valuable input in various drafts revisions of my proposal.

Abstract

The Internet of Things (IoT) offers immense benefits by enabling devices to leverage networked resources thereby making intelligent decisions. The numerous heterogeneous connected devices that exist throughout the IoT system creates new security and privacy concerns. Some of these concerns can be overcome through data trust, transparency, and integrity, which can be achieved with data provenance. Data provenance, also known as data lineage, provides a complete history of transformations that occurs on a data object from the time it was created to its current state. Data provenance has been explored in the areas of scientific computing, business, forensic analysis, and intrusion detection. Data provenance can help in detecting and mitigating malicious cyber attacks.

This dissertation explores the integration of data provenance within the IoT ecosystem by taking an in-depth look at the collection and adoption of provenance in mitigating malicious intrusion. The first step towards provenance adoption is achieved by developing Provenance-Aware IoT (PA-IoT), a provenance collection tool for IoT devices.This framework generates provenance data through application manual instrumentation with low execution overhead. With the integration of PA-IoT, a fundamental problem arises; provenance has the possibility to generate tremendous amount of data. We address the storage challenge of provenance collection by adopting a graph summarization approach which allows grouping of similar graphs into super nodes.

We evaluate the effectiveness of our framework by looking at an application of provenance data using an intrusion detection system, which detects malicious threats against IoT applications.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In recent years,the Internet of Things (IoT) has gathered significant traction which has led to the exponential increase in the number of devices connected to the internet. According to a report released by Cisco [16], it is estimated that a total of 50 million devices will be connected to the internet by the year 2020. Data is generated in an enormous amount in real-time by sensors and actuators contained in these devices. With the vast amounts of connected heterogeneous devices, security and privacy risks are increased. Rapid7 [52], an internet security and analytics organization, released a report highlighting vulnerabilities that exist on select IoT devices. In their report, they outline vulnerabilities in baby monitors which allowed intruders unauthorized access to devices whereby a malicious intruder can view live feeds from a remote location. With provenance information, we can generate an activity trail which can be further analyzed to determine who, where, and how, a malicious attack occurred in order to provide preventive measures to eradicate future or current attacks. [15].

In an IoT system, most of the interconnected heterogeneous devices (things) are embedded systems which require lightweight and efficient solutions as compared to general purpose systems. This requirement is attributed to the constrained memory and computing power of such devices. The vast amount of data generated from IoT devices requires stronger levels of trust which can be achieved through data provenance. Provenance is used to determine causality and effect of operations performed on data objects [20]. Data provenance ensures authenticity, integrity and transparency between information disseminated

across an IoT system. This level of transparency can be translated to various levels across the IoT architectural stack. To achieve transparency in an IoT framework, it is imperative that data provenance and IoT systems be unified to create a provenance aware system that provides detailed records of all data transactions performed on devices connected in an IoT network.

Most provenance-aware systems collect a fixed kind of provenance data (e.g system calls, files, and process) [29, 5, 20]. There is a need to create provenance-aware systems which allows for the flexibility in specifying the kinds of provenance data to collect. Collecting fixed kinds of provenance leads to an increase in the amount of noisy provenance data (i.e. unrelated provenance data which is not required by an organization). By doing this, we provide pruning capabilities [12] of provenance data. Policy specification and implementation details are discussed in chapter 4.

1.2 Provenance-Aware IoT Device Use Case

Consider a smart home as illustrated in Figure 2.1 that contains interconnected devices such as a thermostat which automatically detects and regulates the temperature of a room based on prior information of a user's temperature preferences, a smart lock system that can be controlled remotely and informs a user via short messaging when the door has been opened or is closed, a home security camera monitoring system, a smart fridge which sends a reminder when food products are low. In an event that a malicious intruder attempts to gain access to the smart lock system and security camera remotely, provenance information can be used to track the series of events to determine where and how a malicious attack originated. Provenance can also be used as a safeguard to alert of a possible remote or insider compromise thereby protecting against future or ongoing malicious attacks.

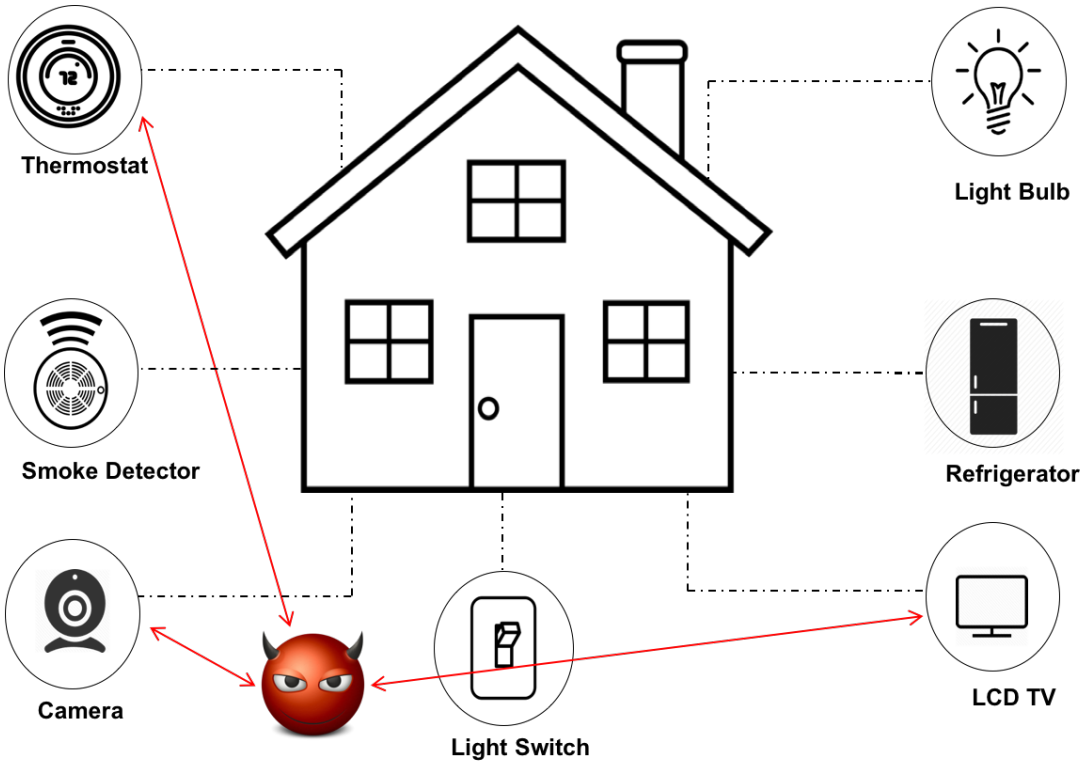

Figure 1.1: Smart home use case Diagram

1.3 Research Questions

The unification of data provenance and IoT is essential to the security of data disseminated in an IoT system. However, the unification raises some important research issues some of which are as a result of pre-existing provenance and IoT related issues. Some of the issues raised as a result of the unification of data provenance with IoT are outlined below:

- **Modeling provenance data:** How do we model provenance data collected from sensors and actuators contained in IoT architecture? Are there models used to represent causality between sensor and actuator readings in the IoT architecture.
- **Memory constraints on IoT Devices:** The vast amounts of data generated leads to high storage space utilization. Memory and data management techniques should

be employed for efficient storage of provenance data on memory constrained devices. How do we effectively collect provenance data in resource constrained devices and relate this information across layers of the IoT architecture

- **Provenance data application utilization:** How do we utilize provenance data generated by provenance collection system to provide intrusion detection or digital forensics.

1.4 Research Contribution

This dissertation explores the integration of data provenance the IoT ecosystem by making the following key contributions:

- **Provenance-Aware Framework for IoT Device.** A provenance collection framework is important for detecting malicious intrusion by depicting causality and data dependency between entities contained in an IoT ecosystem. By depicting data causality and data dependency,we are able to track changes in information flow of sensor events.This system creates groundwork for capturing and storing provenance data across the IoT architectural stack.

- **Efficient provenance storage using graph summarization.** Prior work has shown that provenance collection frameworks generate immense amounts of provenance data which sometime outgrows data in which provenance is generated for. To address this challenge, an efficient provenance storage approach on IoT devices is developed using graph summarization.

- **Evaluating provenance-aware system using graph based intrusion detection.** Evaluation of provenance collection framework is performed by developing an intrusion detection system which heavily relies on provenance data generated by the provenance collection framework. Evaluation is performed on the graph based intrusion detection system using simulated IoT application: One is a simulated heating

air conditioning and ventilation (HVAC) application. The other is an automotive network dataset.

1.5 Organization of Dissertation

The remaining portion of this dissertation is organized as follows: Chapter 2 reviews background information on data provenance and the Internet of Things. It also discusses data provenance model (PROV-DM) for representing provenance. Chapter 3 outlines related work on provenance collection systems. Chapter 4 proposes a provenance collection system for automatic collection of provenance data in an IoT device. It also discusses provenance-sensor model, provenance collection framework's system implementation and evaluation. Chapter 5 describes our framework for efficient provenance storage of provenance data using graph summarization. Chapter 6 concludes with proposed future research directions.

Chapter 2

Background

This chapter describes key concepts of data provenance, IoT characteristics, and provenance models. It outlines the tradeoffs that exists between provenance, log data and metadata.

2.1 Data Provenance

The Oxford English dictionary defines provenance [48] as "the place of origin or earliest known history of something". An example of provenance can be seen with a college transcript. A transcript is the provenance of a college degree because it outlines all of the courses satisfied in order to attain the degree.

2.2 Provenance-Aware IoT Device Use Case

In the field of computing, data provenance, also known as data lineage, can be defined as the history of all activities performed on entities from its creation to its current state. Cheney et al. [15] decribes provenance as the origin and history of data from its lifecycle.

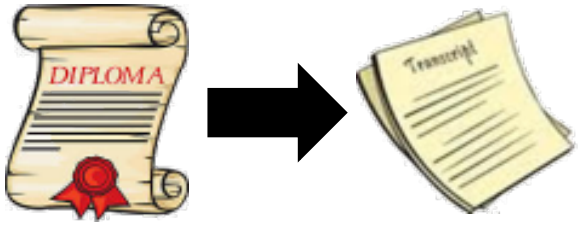

Figure 2.1: Provenance depicted as as a college transcript.

Buneman et al [13] describes provenance from a database perspective as the origin of data and the steps in which it is derived in the database system. We formally define provenance as follows:

Definition 1 *Data provenance of an entity is a comprehensive history of activities that occur on that entity from its creation to its present state.*

Provenance involves two granularity levels: fine-grained provenance and coarse-grained provenance. Fine-grained provenance [20] entails the collection of a detailed level of provenance. Coarse grained provenance, on the other hand, is a collection of a summarized version of provenance. Data provenance has immense applications and has been explored in the areas of scientific computing [22, 5] to track how an experiments are produced, in business to determine the work-flow of a process, and in computer security for forensic analysis and intrusion detection [9, 40, 38] . Provenance denotes the who, where and why of data [15]. An example of provenance for a software system is a web server's log file. This file contains metadata for various request and response time and the ip address of all host systems that requests information from the server. This log file identifies the data object (i.e. Web server), transformations that occurs on this object (e.g read write connections) and the state of the data object. Provenance data is represented as an acyclic graph which denotes casual relationship and dependencies between entities.

Provenance ensures trust and integrity of data [11]. It outlines causality and dependency between all objects involved in the system and allows for the verification of the source of data. Causality and dependency are used to determine the relationship between multiple objects. The relationship in which provenance denotes can in turn be used in digital forensics [58] to investigate the cause of a malicious attack and also in intrusion detection systems to further enhance the security of computing devices.

2.2.1 Provenance Characteristics

Since provenance denotes the who, where and why of data transformation, it is imperative that data disseminated in an IoT architecture satisfies the required conditions. The characteristics of data provenance are outlined in detail below.

- Who: This characteristic provides information on activities made to an entity. Knowing the "who" characteristic is essential because it maps the identity of modification to a particular data object. An example of "who" in an IoT use case is a sensor device identifier.

- Where: This characteristic denotes location information in which data transformation was made. This provenance characteristic could be optional since not every data modification contains location details.

- When: This characteristic denotes the time information at which data transformation occurred. This is an essential provenance characteristic. Being able to tell the time of a data transformation allows for tracing data irregularities.

- What: This characteristic denotes the transformation is applied on a data object. A use case for IoT can be seen in the various operations (create, read, update, and delete) that could be performed on a file object.

2.3 Comparing Provenance with Log Data and Metadata

Provenance data, log data and metadata are key data concepts that often are used interchangeably. We try to address the differences and similarities between provenance data, metadata and log data.

2.3.1 Provenance and Metadata

Metadata and provenance are often considered related but yet subtle distinctions exist. Metadata contains descriptive information about data. Metadata can be considered as provenance when there exists a relationship between objects and they explain the transformation that occurs. In summary, metadata and provenance are not the same, however an overlap exists. Metadata contains valuable provenance information but not all metadata is provenance information.

2.3.2 Provenance and Log data

Log data contains information about the activities of an operating system or processes. Log data can be used as provenance because It contains data trace specific to an application domain. Log files might contain unrelated information such as error messages, warnings which might not be considered as provenance data. Provenance allows for specified collection of information that relates to the change of the internal state of an object. In summary, log data could provide insight to what provenance data to collect.

2.4 Internet of Things (IoT)

There is no standard definition for IoT, however, researchers have tried to define the concept of connected "things". The concept of IoT was proposed by Mark Weiser in the early 1990s [37] which represents a way in which the physical objects, "things", can be connected to the digital world. Gubbi et al [45] defines the IoT as an interconnection of sensing and actuating devices that allows data sharing across platforms through a centralized framework. We define (IoT) as follows:

Definition 2 *The Internet of Things (IoT) is a network of heterogeneous devices with sensing and actuating capabilities communicating over the internet.*

The notion of IoT has been attributed to smart devices. The interconnectivity between various heterogeneous devices allows for devices to share information in a unique manner. Analytics is a driving force for IoT. With analytics, devices can learn from user data to make smarter decisions. This notion of smart devices is seen in various commercial applications such as smartwatches, thermostats that automatically learns a user patterns. The ubiquitous nature of these devices make them ideal choices to be included in consumer products. IoT architecture consists of four distinct layers: The sensor and actuator layer, device layer, gateway layer and the cloud layer.

With the recent data explosion [28] due to the large influx in amounts of interconnected devices, information is disseminated at a fast rate and with this increase involves security and privacy concerns. Creating a provenance-aware system is beneficial to IoT because it ensures the trust and integrity of interconnected devices. Enabling provenance collection in IoT devices allows these devices to capture valuable information which enables backtracking in an event of a malicious attack. We take a holistic approach to provenance collection by looking at how provenance information is collected across an IoT architectural framework.

2.4.1 IoT Architecture

IoT architecture represents a functional hierarchy of how information is disseminated across multiple hierarchies contained in an IoT framework; from devices which contain sensing and actuating capabilities to massive data centers (cloud storage). Knowing how information is transmitted across layers allows a better understanding on how to model the flow of information across actors contained in an IoT hierarchy.

Figure 2.2 displays the IoT architecture and the interactions between the respective layers. The base of the architectural stack consist of sensors and actuators which gathers provenance information and interacts with the device layer. The device layer consists of devices (e.g mobile phones, laptops, smart devices) which are responsible for aggregating data collected from sensors and actuators. These devices in turn forwards the aggregated data to the gateway layer. The gateway layer routes and forwards data collected from the

device later. It could also serve as a medium of temporary storage and data processing. The cloud layer is involved with the storage and processing of data collected from the gateway layer. Note that the resource constraints decreases up the architectural stack with the cloud layer having the most resources (memory, power computation) and the sensor-actuator layer having the least.

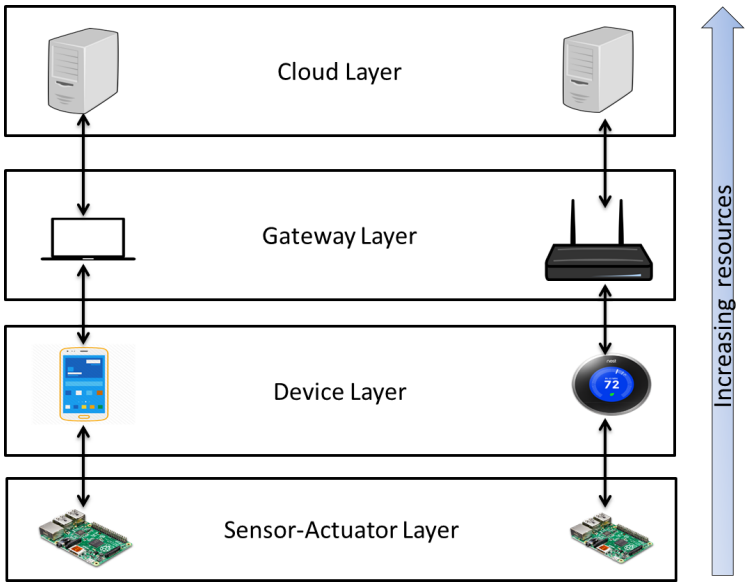

Figure 2.2: IoT Architecture Diagram. The arrows illustrates the interaction between data at various layers on the architecture.

2.5 IoT Application Use Case

2.6 Model for representing data provenance

In order to represent provenance in an IoT ecosystem, we need to satisfy provenance characteristics (i.e who, where, when, and what of data transformations).

Provenance of sensor readings in an IoT device should describe the dependency relationships between all entities responsible for producing those readings. We adopt the

Provenance Data Model (PROV-DM) [1], a W3C standard which conforms to Provenance Ontology (PROV-O) and is used to depict dependencies between entities, activities and agents (digital or physical). PROV-DM creates a common model that allows for interchange of provenance information between heterogeneous devices and is represented serialized in three formats: XML, JSON and RDF.

PROV-DM contains two major components: types and relations. Types can be entities, activities, or agents. An entity is a physical or digital object. An activity represents some form of action that occurs over time. An agent takes ownership of an entity, or performs an activity. Figure 2.3 illustrates the types and relations contained in PROV-DM and their graphical representation. Entities, activities and agents are represented as oval, rectangular and pentagonal shapes respectively.

We propose an alignment to the PROV-DM which contains information such as sensor readings, device name, and device information and give a detailed description of the alignment with use cases. Details on the provenance-sensor data model alignment is discussed in section IV.

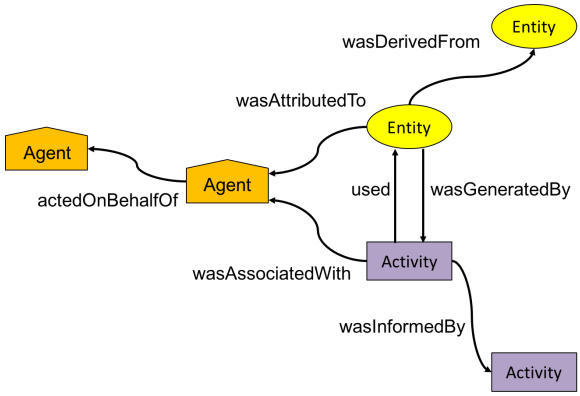

Figure 2.3: Prov-DM respresentation showing types in the model (Entity, Activity, and Agent) and the relationships between them

PROV-DM defines the following seven relationships between the types.

- wasGeneratedBy: Signifies the production of a new entity by an activity.

- used: An entity generated by one activity has been adopted by another activity.
- wasInformedBy: Signifies the exchange of an entity by two activities.
- wasDerivedFrom: Represents a copy of information from an entity.
- wasAttributedTo: Denotes relational dependency between an entity and an agent when the activity that created the agent is unknown.
- wasAssociatedWith: An agent created or modified the activity
- actedOnBehalfOf: Delegation of authority from an agent to itself or another agent to perform a particular responsibility.

2.6.1 PROV-JSON

PROV-JSON is used for representing PROV-DM data in JSON (JavaScript Object Notation) format. It contains all of the components and relationships contained in PROV-DM and allows for easy serialization and deserialization. JSON is a lightweight data format which is human readable and easy to parse. Figure 2.4 illustrates a use case for the serialization of PROV-DM to PROV-JSON. PROV-JSON contains key-value pairs and can be considered as an indexed version of PROV-DM. The example illustrates the relationship in which s1 tries to access a file (FileB) which was generated by s2. PROV-JSON contains an entity, activity and agent type which are represented as a json objects and are identified by their respective ids. Identifiers are optional in PROV-DM but are required for PROV-JSON since json objects contains key-value pairs, the id's of each object has to be implicitly specified and cannot contain null values.

Each object contains fields which assigns additional attributes to the relations (i.e name, type, prov:startTime). PROV-JSON documents might also contain a prefix object which defines namspaces that are used in the document. In this object is contained a default field which is used to define all of the namespace that contains all other unprefixed namespace.

```
{
     ...
    "entity":{
        "e1": {
        "ex:name": "FileB"
        }
    },
    "agent":{
        "a1": {

            "ex:name": "s1",
            "prov:type": "sensor"

         }
        ...
    },
     "agent":{
        "a2": {
            "name": "s2",
            "ex:type": "sensor"
         }
        ...
    },
    "activity":{
        "activity1": {
              "prov:startTime": "2011-11-16T16:05:00",
              "prov:endTime": "2011-11-16T16:06:00",
              "prov:type": "read"
        }
      ...
    },
     "wasGeneratedBy":{
            "wGB1": {
                "prov:entity": "e1",
                "prov:activity": "a2",
                "prov:time": "2011-11-16T16:00:00"
          }
         ...
    },
          "used":{
            "u1": {
                "prov:entity": "a1",
                "prov:activity": "e1",
                "prov:time": "2011-11-16T16:00:00"
          }
         ...
    }
}
```

Figure 2.4: PROV-JSON MODEL ©[2]

Chapter 3

Related Work on Provenance Collection Systems

This section discusses the state of the art in the area of data provenance collection systems, data compression techniques for data provenance, and models for representing provenance data.

3.1 Related Work on Provenance Collection Systems

There has been a considerable amount of work done on data provenance collection [35, 10, 18, 40]. Some of the work done has been focused on databases, sensor networks, scientific workflow systems and file system provenance but so far little attention has been given to provenance in IoT. Some of the prior work done on data provenance collection are outlined below:

3.1.1 Provenance Aware Storage System (PASS)

Muniswamy Reddy et al [38] developed a provenance collection system that tracks system-level provenance of the Linux file system. Provenance information is stored in the same location as the file system for easy accessibility, backup, restoration, and data management. Provenance information is collected and stored in the kernel space. PASS is composed of 3 major components: provenance collector, provenance storage,and provenance query. The collector keeps track of system level provenance. It intercepts system calls which are translated into provenance data and initially stored in memory as inode cache. Provenance

data is then transferred to a file system in an kernel database, BerkleyDB. This database maps key value pairs for provenance data for fast index look up. PASS collects and stores provenance information containing a reference to the executable that created the provenance data, input files, a description of hardware in which provenance data is produced, OS information, process environment, process parameters, and other data such as a random number generator seed. PASS detects and eliminates cycles that might occur in provenance dependencies as a result of version avoidance. Cycles violate the dependency relationships between entities. For example, a child node could depend on a parent node and also be an ancestor of a parent node. PASS eliminates cycles by merging processes that might have led to the cycles. It also provides functionality for querying provenance data in the database. The query tool is built on top of BerkleyDB. For querying provenance, users can process queries using the provenance explorer. Query is done using commands such as MAKEFILE GENERATION which creates the sequence of events that led to the final state of a file or process. DUMP ALL, gets the provenance of the requested file.

Our approach looks at data provenance with data pruning as a key requirement since we are dealing with devices with limited memory and storage capabilities. We employ a policy based approach which allows provenance pruning by storing what provenance data is considered important to a specific organization. Also,unlike PASS, our system collects not just system level provenance but also application level provenance.

3.1.2 HiFi

Bates et al. [47] developed system level provenance collection framework for the Linux kernel using Linux Provenance Modules(LPM), this framework tracks system level provenance such as interprocess communication, networking, and kernel activities. This is achieved by mediating access to kernel objects using Linux Security Model(LSM) which is a framework that was designed for providing custom access control into the Linux kernel. It consists of a set of hooks which executed before access decision is made. LSM was designed to avoid problem created by direct system call interception. The provenance information collected

from the kernel space is securely transmitted to the provenance recorder in the user space.

HiFi contains three components: provenance collector, provenance log and provenance handler. The collector and log are contained in the kernel space while the handler is contained in the user space. The log is a storage medium which transmits the provenance data to the user space. The collector uses LSM which resides in the kernel space. The collector records provenance data and writes it to the provenance log. The handler reads the provenance record from the log. This approach to collecting provenance data differs from our work since we focus on embedded systems and are concerned with input and output (I/O) data, which involves sensor and actuator readings. Additionally, HiFi deals with collecting system level events which might incur additional overhead when compared to collecting application level provenance. HiFi is engineered to work solely on the Linux operating system. Embedded systems that do not run on Linux OS will not be able to incorporate HiFi.

3.1.3 RecProv

RecProv [27] is a provenance system which records user-level provenance, avoiding the overhead incurred by kernel level provenance recording. It does not require changes to the kernel like most provenance monitoring systems. It uses Mozilla rr to perform deterministic record and replay by monitoring system calls and non deterministic input. Mozilla rr is a debugging tool for Linux browser. It is developed for the deterministic recording and replaying of the Firefox browser in Linux. Recprov uses PTRACE_PEEKDATA from ptrace to access the dereferenced address of the traced process from the registers. Mozilla rr relies on ptrace, which intercepts system calls to monitor the CPU state during and after a system call. It ptrace to access the dereferenced address of the traced process from the registers. System calls are monitored for file versioning. The provenance information generated is converted into PROV-JSON, and stored in Neo4j, a graph database for visualization and storage of provenance graphs.

3.1.4 StoryBook

Spillance et al [51] developed a user space provenance collection system, Storybook, which allows for the collection of provenance data from the user space thereby reducing performance overhead. This system takes a modular approach that allows the use of application specific extensions allowing additions such as database provenance, system provenance, and web and email servers. It achieves provenance capture intercepting system level events on FUSE, a file system and MySQL , a relational database. StoryBook allows developers to implement provenance inspectors these are custom provenance models which captures the provenance of specific applications which are often modified by different application(e.g web servers, databases). When an operation is performed on a data object, the appropriate provenance model is triggered and provenance data for that data object is captured. StoryBook stores provenance information such as open, close, read or write, application specific provenance, and causality relationship between entities contained in the provenance system. Provenance data is stored in key value pairs using Stasis and Berkely DB as the storage backend. It also allows the use of interoperable data specifications such as RDF to transfer data between various applications. Storybook allows for provenance query by looking up an inode in the ino hashtable. Collecting application level and kernel level events is similar to our approach of provenance collection, however, our approach integrates the use of a policy to eliminate noisy provenance data thereby allowing only relevant provenance to be stored.

3.1.5 Trustworthy Whole System Provenance for Linux Kernel

Bates et al [9] provide a security model for provenance collection in the linux kernel. This is achieved by creating a Linux Provenance Model (LPM). LPM serves as a security layer which provides a security abstraction for provenance data. It is involved in the attestation disclosure of the application layer and authentication channel for the network layer. The goal of LPM is to provide an end to end provenance capture system. LPM ensures

the following security guarantees: For LPM,the system must be able to detect and avoid malicious forgery of provenance data. The system must be trusted and easily verifiable.

When an application invokes a system call, the LPM tracks system level provenance which is transmitted to the provenance module via a buffer. The provenance module registers contain hooks which records system call events. These events are sent to a provenance recorder in the user space. The provenance recorder ensures that the provenance data is stored in the appropriate backend of choice. Provenance recorders offer storage support for Gzip, PostGreSQL, Neo4j and SNAP.

3.1.6 Towards Automated Collection of Application-Level Data Provenance

Tariq et al. [53] all developed a provenance collection system which automatically collects interprocess provenance of applications source code at run time. This takes a different approach from system level provenance capture. It achieves provenance capture by using LLVM compiler framework and SPADE provenance management system. LLVM is a framework that allows dynamic compilation techniques of applications written in C, C++, Objective-C, and Java. Provenance data is collected from function entry and exit calls and is inserted during compilation. LLVM contains an LLVM reporter. This is a Java class that parses the output file collected from the LLVM reporter and forwards the provenance data collected to the SPADE tracer. SPADE is a provenance management tool that allows for the transformation of domain specific activity in provenance data. The LLVM Tracer module tracks provenance at the exit and entry of each function. The output is an Open Provenance Model in which functions are represented by an OPM process, arguments and return values are represented as an OPM artifact. To minimize provenance record, users are allowed to specify what functions they would like to collect provenance information at run time. The LLVM optimizer generates a graph of the target application. This graph is used to perform reverse reachability analysis from the functions contained in the graph. A

workflow of how the framework works in collecting provenance is as follows:

- The application is compiled and converted into bitcode using the LLVM C compiler, clang.
- The LLVM Tracer module is compiled as a library.
- LLVM is run in combination with the Tracer, passed to include provenance data.
- Instrumentation bitcode is converted into assembly code via the compiler llc
- The socket code is compiled into assembly code.
- The instrumented application and assembly code is linked into an executable binary and sent to the SPADE kernel.

One major limitation to this approach of collecting application level provenance is that a user is required to have the source code of the specific application in which provenance data is required. Also, provenance collection is limited to function's exit and entry points. Provenance collection deals with data pruning by allowing a user to specify what provenance data to collect which is similar to our policy based approach for efficient provenance data storage.

3.1.7 Provenance from Log Files: a BigData Problem

Goshal et al [19] developed a framework for collecting provenance data from log files. They argue that log data contains vital information about a system and can be an important source of provenance information. Provenance is categorized based on two types: process provenance and data provenance. Process provenance involves collecting provenance information of the process in which provenance is captured. Data provenance on the other hand describes the history of data involved in the execution of a process. Provenance is collected by using a rule based approach which consists of a set of rules defined in XML.

The framework consists of three major components. The rule engine which contains XML specifications for selecting, linking and remapping provenance objects from log files. The rule engine processes raw log files to structured provenance. There are three categories of rules as specified by the grammar. The match-set rules which selects provenance data based on a set of matching rules. Link rules which specifies relationship between provenance events and remap rules are used to specify an alias for a provenance object. The rule engine is integrated with the log processor. The log processor component is involved with parsing log files with information contained in the rule engine. It converts every matching log information to a provenance graph representing entities (vertices) and their relationship (edges). The adapter converts the structured provenance generated by the log processor into serialized XML format which is compatible with Karma, a provenance service application that is employed for the storage and querying of the provenance data collected.

3.1.8 Towards a Universal Data Provenance Framework using Dynamic Instrumentation

Gessiou et al [18] propose a dynamic instrumentation framework for data provenance collection that is universal and does not incur the overhead of kernel or source code modifications. This is achieved by using DTrace, a dynamic instrumentation utility that allows for the instrumentation of user-level and kernel-level code and with no overhead cost when disabled.

The goal is to provide an easy extension to add provenance collection on any application regardless of its size or complexity. Provenance collection is implemented on the file system using PASS and evaluated on a database(SQLite) and a web browser(Safari). The logging component monitors all system calls for processes involved. The logging component contains information such as system-call arguments, return value, user id, process name,and timestamp. The logging components includes functionalities to collect library

and functional calls. The authors argue that applying data provenance to different layers of the software stack can provide varying levels of information. Provenance needs to be captured at the layer in which it is closest to the information that is needed to be captured. Provenance information that pertains to complex system activities such as a web browser or a database are collected. The information is collected from a process system-call and functional-call based on valid entry points contained in the process. DTrace dynamic instrumentation helps in discovering interesting paths in a system in which data propagates. This helps users use this information to collect a more accurate provenance data.

3.1.9 Provenance-Based Trustworthiness Assessment in Sensor Networks

Lim et al. [34] developed a model for calculating the trust of nodes contained in a sensor network by using data provenance and data similarity as deciding factors to calculate trust. The value of provenance signifies that the more similar a data value is, the higher the trust score. Also, the more the provenance of similar values differ, the higher their trust score. The trust score of a system is affected by the trust score of the sensor that forwards data to the system. Provenance is determined by the path in which data travels through the sensor network. This work differs from our approach since the authors focus on creating a trust score of nodes connected in a sensor network using data provenance and do not emphasize how the provenance data is collected. We are focused on creating a provenance aware system for sensor-actuator I/O operations which may be used to ensure trust of connected devices.

3.1.10 Backtracking Intrusions

Samuel et al [29] developed a system, Backtracker, which generates an information flow of OS objects (e.g file, process) and events (e.g system call) in a computer system for the purpose of intrusion detection. From a detection point, information can be traced to pinpoint

where a malicious attack occurred. The information flow represents a dependency graph which illustrates the relationship between system events. Detection point is a point in which an intrusion was discovered. Time is included in the dependency between objects to reduce false dependencies. Time is denoted by an increasing counter. The time is recorded as the difference of when a system call is invoked till when it ends. Backtracker is composed of two major components: EventLogger and GraphGen. An EventLogger is responsible for generating system level event log for applications running on the system. EventLogger is implemented in two ways: using a virtual machine and also as a stand alone system. In a virtual machine, the application and OS is run within a virtual machine. The OS running inside of the virtual machine is known as the guest OS while the OS on the bare-metal machine is known as the host OS, hypervisor or virtual machine monitor. The virtual machine alerts the EventLogger in the event that an application makes a system call and or exits. EventLogger gets event information, object identities, dependency relationship between events from the virtual machine's monitor and also the virtual machine's physical memory. EventLogger stores the collected information as a compressed file. EventLogger can also be implemented as a stand alone system which is incorporated in an operating system. Virtual machine is preferred to a standalone system because of the use of virtual machine allows ReVirt to be leveraged. ReVirt enables the replay of instruction by instruction execution of virtual machines. This allows for whole information capture of workloads. After the information has been captured by the EventLogger, GraphGen is used to produce visualizations that outlines the dependencies between events and objects contained in the system. GraphGen also allows for pruning of data using regular expressions which are used to filter the dependency graph and prioritize important portions of the dependency graph.

3.1.11 Provenance Recording for Services

Grouth et al [22] developed a provenance collection system, PReServ which allows software developers to integrate provenance recording into their applications. They introduce P-assertion recording protocol as a way of monitoring process documentation for Service

Oriented Architecture (SOA) in the standard for grid systems. PReServ is the implementation of P-assertion recording protocol (PreP). PreP specifies how provenance can be recorded. PReServ contains a provenance store web service for recording and querying of P-assertions. P-assertions are provenance data generated by actors about the application execution. It contains information about the messages sent and received as well as state of the message. PReServ is composed of three layers, the message translator which is involved with converting all messages received via HTTP requests to XML format for the provenance store layer. The message translator also determines which plug-in handle to route incoming request to, the Plug-Ins layer. It implements functionality that the Provenance Store component provides. This component allows third party developers to add new functionality to store provenance data without going through details of the code implementation. The backend component stores the P-assertions. Various backend implementations could be attached to the system by the plug-in component.

PReServ was developed to address the needs of specific application domain (Scientific experiments). It deals with recording process documentation of Service Oriented Architecture (SOA) and collects P-assertions which are assertions made about the execution (i.e messages sent and received, state information) of actions by actors. The provenance collected by PReServ does not demonstrate causality and dependency between objects. In other words, the provenance data collected are not represented using a provenance model (e.g PROV-DM) that depicts causal relationship between provenance data.

3.2 Policy-Based Provenance Data Storage

3.2.1 Take Only What You Need: Leveraging Mandatory Access Control Policy to Reduce Provenance Storage Costs

Bates et al [8] developed a system which allows maximizing provenance storage collection in the Linux OS environment using mandatory access control (MAC) policy. This en-

ables avoiding the collection of unnecessary provenance data and only records interesting provenance that is important to an application. In a MAC system, every object contains a label and the MAC policy specifies interactions between different labels. Provenance policy contains security context which is enforced by the MAC policy and contains permissions based on security context. It also provides the connection between provenance and MAC. The authors argue that the system collects complete provenance without gaps in provenance relationship. This is achieved by extending the work of Vijayakumar et al.s [55] Integrity Walls project which allows mining SELinux policies with the aim of finding Minimal Trusted Computing Base (MTCB) of various applications. A policy document is divided into trusted and untrusted labels. The trusted labels provides a thorough description of events that can have access to the application and this information serve as a provenance policy that ensures the complete provenance relationship.

3.2.2 A Provenance-aware policy language (cProvl) and a data traceability model (cProv) for the Cloud

Ali et al [4] provides a provenance model, cProv, which illustrates the relationship between entities contained in a cloud system. This model is build on PROV notation. cProv contains 5 derived nodes(cprov:cProcess, cprov:Resource, cProv:Transition, cprov:cResource, and cprov:pResource). There are a total of 10 edges built on the relationships contained in the PROV notation. Node consists of properties such as location, time-to-live, virtual-to physical mappings, executed operations. They also develop a policy language, cProvl that allows for access of provenance data. cProvl is represented in XML and consists of rules and conditions that expresses logic to enforce access of resources. Each rule contains an entity. XACML, a policy language for the enforcement of access control is used for implementing access control on the provenance policy information. cProvl is mapped to XACML to allow the policy to run on XACML. An example of how the policy structure works is as follows: An application makes a request, this request is converted to an appropriate provenance

aware request. This request is then forwarded to the policy engine. The request is evaluated by this layer by evaluating one or more policy rules as contained in a policy. This creates a response which is forwarded to the provenance converter that converts the request to an appropriate format for client utilization.

3.3 Compressing Provenance Graphs

Xie et al proposed a compression technique which is an extension of web compression.

3.4 Provenance Data Compression

3.4.1 Secure Data Provenance Compression Using Arithmetic Coding in Wireless Sensor Networks (WSN)

Hussein et al. [26] encode the provenance of the path in which sensor provenance travels using arithmetic coding, a data compression algorithm. Arithmetic coding assigns intervals to a string of characters by cumulative probabilities of each character contained in the string. It assign probabilities by monitoring the activities of the WSN, this is known as the training phase. In this phase the number of times a node appears in a packet is counted. Each node in the WSN is regarded as a symbol. This is used as input to the arithmetic coding algorithm which generates a variable interval for the combination of symbols. The WSN contains a Base Station (BS) and a network of nodes. The BS is in charge of verifying the integrity of the packets sent. It is also where the encoded characters using arithmetic coding are decoded. For their application, provenance is referred to as the path in which data travels to the BS. Here, provenance is divided into two types: Simple provenance in which data generated from the leaf nodes, are forwarded to surrounding nodes towards the BS. In the other form of provenance, data is aggregated at an aggregation node and forwarded to a BS. Each node in the WSN is represented as a string of characters and is

encoded using arithmetic coding. The BS is responsible for decoding encoded provenance information. Provenance is encoded at the respective nodes and forwarded to the next node in the sensor network. The BS recovers characters the same way they were encoded. It also receives the encoded interval to which it uses to decode characters by using the probability values saved in its storage.

One limitation to their approach of provenance pruning is that provenance is considered as the path in which data travels to get to the base station and not the data that is transmitted itself. Provenance data collected are sensor ids which are represented as single characters. In contrast to their approach, our provenance collection framework collects provenance data not just pertaining to the identification of the "who" provenance characteristics but also other components of provenance. We also address data pruning by using a policy based approach in which a policy is specified for what kinds of provenance data to collect thereby eliminating noisy data.

Chapter 4

Sensor Data Provenance Collection Framework for the IoT

In this chapter, we explore the integration of provenance within the IoT. We introduce Provenance Aware Internet of Things System (PAIoTS), a provenance collection framework for IoT devices. We evaluate the effectiveness of our framework by developing a prototype system for proof of concept.

4.1 Introduction

The Internet of Things (IoT) is transforming home, industrial and commercial automation exponentially and increasing the number of devices connected to the Internet. Cisco estimates that over 50 million devices will be connected to the internet by 2020 [16]. With the increasing amounts of connected heterogeneous devices, security and privacy risks also increase. For example, vulnerabilities in a brand of baby monitors allowed unauthorized access to devices whereby a malicious intruder can view live feeds from a remote location [52].

Due to the heterogeneity of devices and and the sensitivity of data generated on IoT devices, trust is a critical step to ensuring the security of IoT devices. This can be achieved through data provenance which is a comprehensive history of activities that occurred on an entity from its origin to its current state. Provenance ensures confidence in the fidelity of data. Provenance has been applied in domains such as scientific workflows for experiment reproducibility, information security as a form of access control, and also for

intrusion detection. For IoT devices (things) that produce lots of sensor-actuator data, a workflow representation of sensor data can depict dependency between sensor readings and information stored or transmitted by the device.

In this paper, we introduce Provenance Aware Internet of Things System (PAIoTS), a provenance collection framework for IoT devices, in which provenance data is collected and modeled to represent dependencies between sensor-actuator readings and IoT entities. Most of the interconnected heterogeneous devices are embedded systems that require lightweight and resource-efficient solutions as compared to general purpose systems. This requirement is attributed to the constrained memory and computing power of such devices. We also contribute a provenance sensor model which provides a means to convert sensor event traces to provenance model graphs.

4.2 Provenance-Sensor Model

In this section, we introduce the Provenance-Sensor Model and explain how PROV-O is used to convert sensor readings to provenance. Sensor data contains observation information such as temperature, and location details which can be transformed to standardized data interchange formats (RDF, XML, JSON). Sensor data are time series data which can be traced over time. Trace data containing sensor readings are important but do not depict dependency relationships when used alone. We transform trace data to provenance to represent causality and dependency relationships between entities in an IoT system. Provenance can be represented as a directed acyclic graph and the edge between two entities is considered a relation. Relations between data objects follows provenance ontology which depicts transformation between entities. We integrate PROV-O and sensor data for better representation of dependency relationships between trace data generated.

A single sensor might posses the ability to collect multiple trace data, td. For example, a sensor might be able to collect sensor readings of temperature, location, humidity. A combination of trace data at a particular point in time is considered an event. We define

an event for sensor s_1 at time t as $e = \{td_1, ...td_n\}$ where td_1 is the first trace data collected by s_1 and td_n is the last which occurs at time t.

Adopting provenance ontology to IoT, we represent device information as agents (prov:agents), the operation performed on sensor readings (read, create, update) as a provenance activity (prov:activity), and events as entities (prov:entity). A sensor trace is defined as a tuple (t, e, a, s_1, r_1) where t represents a timestamp, e an event, a an operation, s_1 sensor information and r_1 device information.

Device with one sensor

Consider a humidity and temperature sensor s_1, connected to device r_1, a Raspberry Pi. Event $e = \{temperature, humidity\}$ therefore the tuple representation of trace data for sensor s_1 at time t_1 is $(t_1, \{temperature, humidity\}, a, s_1, r_1)$. a is the operation performed on the sensor. Each tuple is mapped to the Provenance Ontology representation using the defined constructs in section IV. Since s_1 is contained in device r_1, r_1 forms an edge with s_1 with the used relation. (e.g r_1 used s_1).

Device with multiple sensors

Figure 4.1 further illustrates the concept of mapping provenance to sensor data using graphical representations from PROV-DM relations and types. The figure depicts a device, r_1, connected to three sensors s_1, s_2, and s_3 with events $[e_1, e_2, ..., e_n]$. Data is generated by three identical temperature sensors, s_1, s_2 and s_3. The graph represents data dependency between r_1, the three sensors, the activity performed by the sensors (the sensors generate data in this case) and the events. For each sensor, the total number of tuple is equal to the number of events. For example, sensor trace data from sensor s_1 is represented by four tuples: $(t_1, e_1, create, s_1, r_1)$,$(t_1, e_2, create, s_1, r_1)$, $(t_1, e_3, create, s_1, r_1)$, and $(t_1, e_4, create, s_1, r_1)$. Each event makes an edge with the preceding event. Edges between events are denoted with a dotted arrow. This represents time dependency between events and that each event occurs consecutively at distinct time intervals.

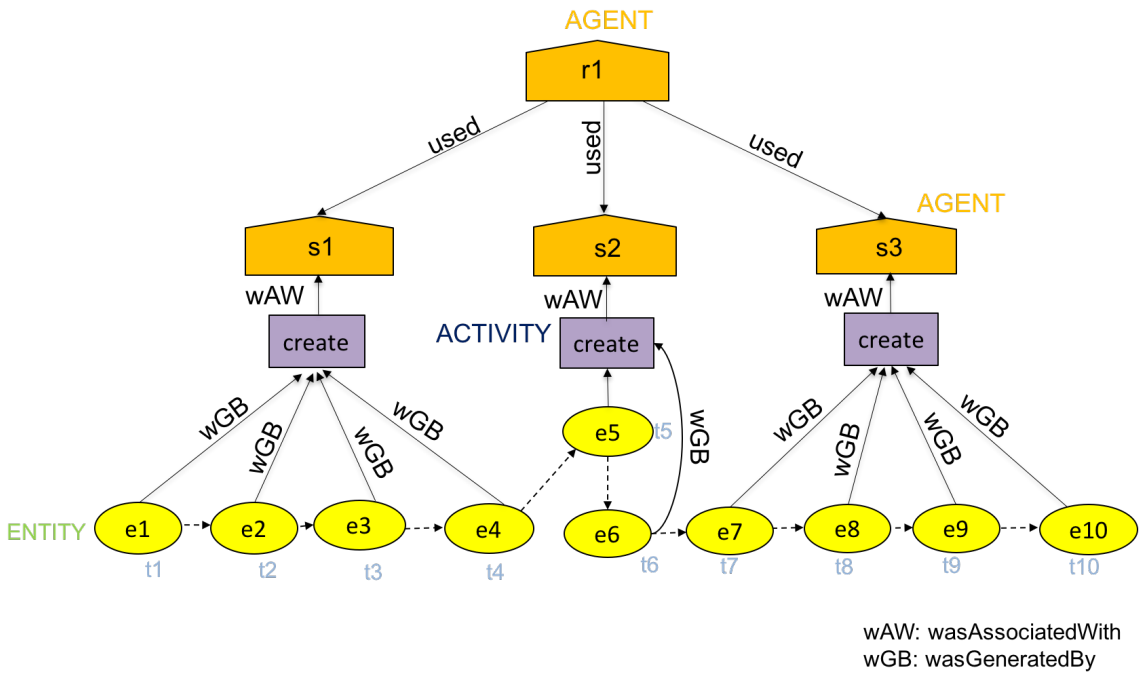

Figure 4.1: Provenance-Sensor Model

Algorithm **??** presents the steps taken by PAIoT to map sensor trace data into graph based provenance. F represents a list of k tuples. For each tuple contained in F, s, and r_1 represents sensor and device information respectively and are defined as agents. e is defined as an event, a is defined as an activity. p is a memory representation of provenance information containing all provenance types and their relations. x and y are a list of relations between sensor-device and activity-sensor, respectively.

4.3 PAIoTS System Implementation

In this section, we outline PAIoTS, a trace-based provenance collection system for IoT devices. Figure 4.3 displays the system architecture. Sensor readings in the form of input and output (I/O) events are recorded by the tracer component. This component intercepts I/O and produces trace information represented in Common Trace Format (CTF).

Algorithm 1 Provenance-Sensor Mapping

```
1: procedure TRACE2PROV(F)
2:     p ← createProvDocument()
3:     for k ∈ F do
4:         if s ∉ p then
5:             s ← createAgent()
6:         end if
7:         if r_1 ∉ p then
8:             r_1 ← createAgent()
9:         end if
10:        e ← createEntity()
11:        a ← createActivity()
12:        if x ∉ p then
13:            x ← relateSensorToDevice()
14:        end if
15:        if y ∉ p then
16:            y ← relateActivityToSensor()
17:        end if
18:        z ← relateEntityToActivity()
19:    end for
20:    return p
21: end procedure
```

PAIoTS converts CTF trace data to provenance in our Provenance-Sensor Model. This conversion can happen at any layer of the IoT stack. CTF conversion to PROV-DM is done using babeltrace. Babeltrace is a plugin framework which allows the conversion of CTF traces into other formats. Trace or provenance data is securely transmitted to a gateway and later transmitted and stored in a cloud backend. Our backend of choice is Neo4j, a graph database with support for efficient storage, query and visualization of provenance data.

CTF contains a mandatory stream known as metadata. Metadata contains information about other streams. It allows parsing a stream without specifying a layout. CTF encodes binary trace output information containing multiple streams of binary events such as I/O activity. Each event can be broken into streams. Streams allow for fast processing since they do not have to be stored in disk before being sent over a network or processed in memory.

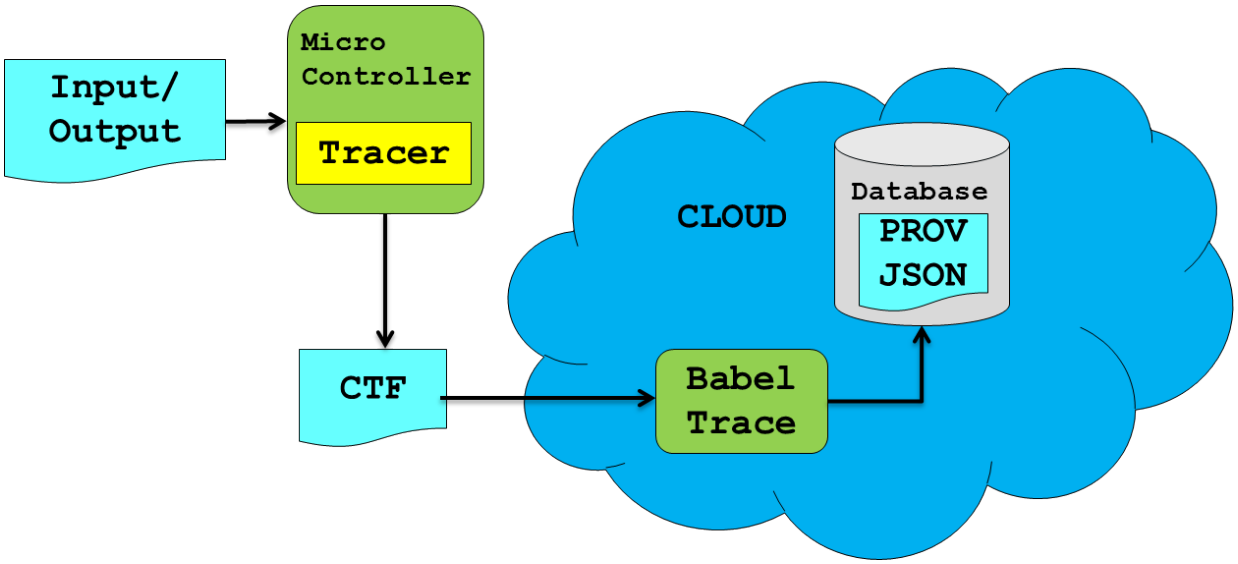

Figure 4.2: System Architecture for PAIoTS.

Our provenance collection system records transformations of I/O data for devices connected in the IoT. For our implementation, we use several tools and hardware components

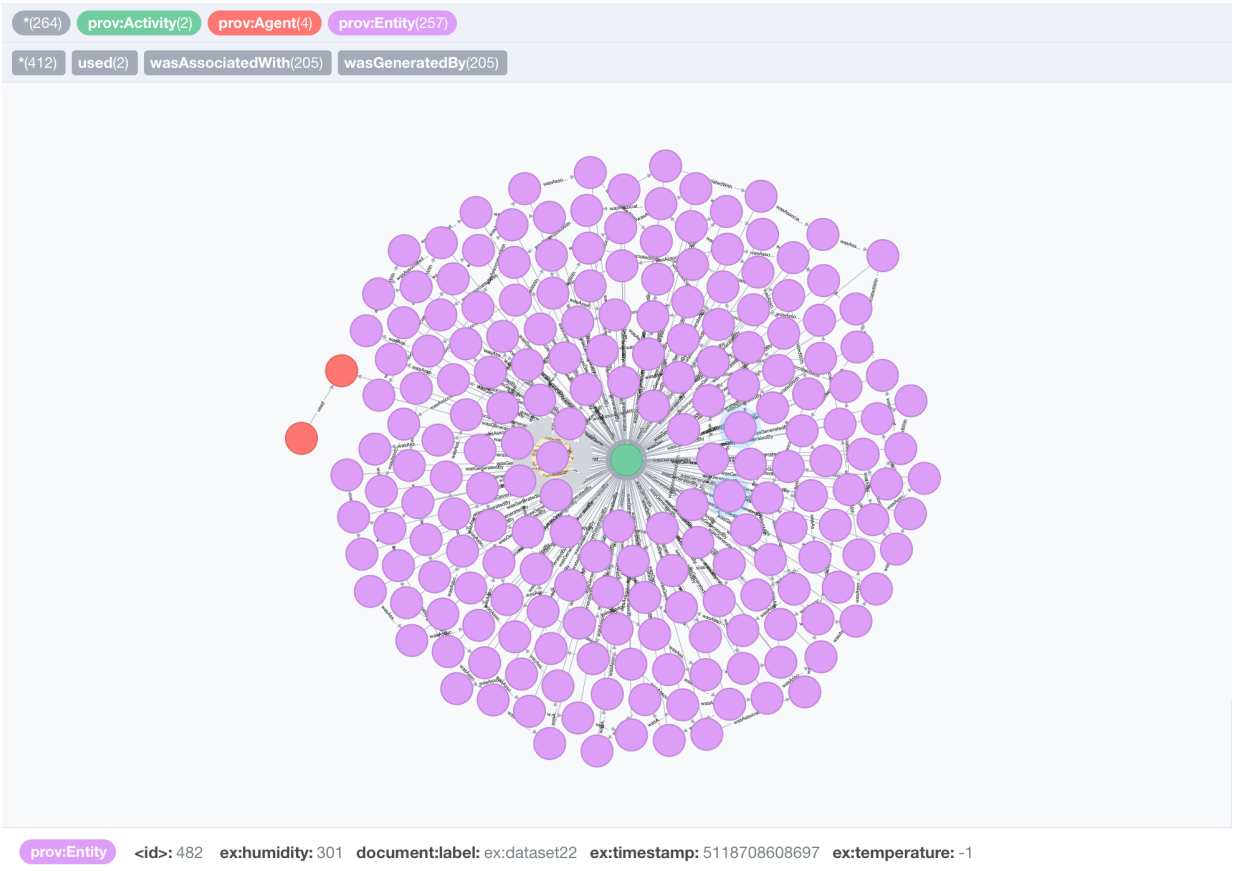

Figure 4.3: Neo4j graph.

in the development of our prototype:

- Raspberry Pi is the microcontroller used to demonstrate our approach. We chose Raspberry Pi because it is a representation of what can be found on an IoT gateway device. Raspberry Pi is a low cost, simple IoT demonstrator.
- Neo4j is a graph database which allows optimized querying of graph data such as provenance.
- Babeltrace is a trace converter tool to convert traces from one format into another.
- barectf is a light weight generator of C code that generates trace data in CTF.

- A yaml generator in barectf creates yaml configuration files with information babeltrace needs to generate CTF trace output. Configuration files contain settings such as an application trace stream, packet type, payload type and size.

4.4 Experiment Evaluation

To evaluate the effectiveness of PA-IoT, we calculated the runtime overhead incurred by comparing execution time of a sample application running without our PA-IoT (this serves as our baseline) and the same application running with PA-IoT instrumented in the application source's code. Table 4.1 displays the result of the comparison. From the results, PA-IoT includes a time execution overhead of 1.45%. We can conclude that our framework does not incur a significant overhead.

Table 4.1: Execution time overhead (in seconds)

Baseline	PA-IoT	Overhead (%)
3.81	3.87	1.57

4.5 Summary

In this chapter, we motivate the need for integrating provenance into the IoT system. We propose PAIoTS, a lightweight provenance collection framework that provides provenance collection capabilities for devices in an IoT system.

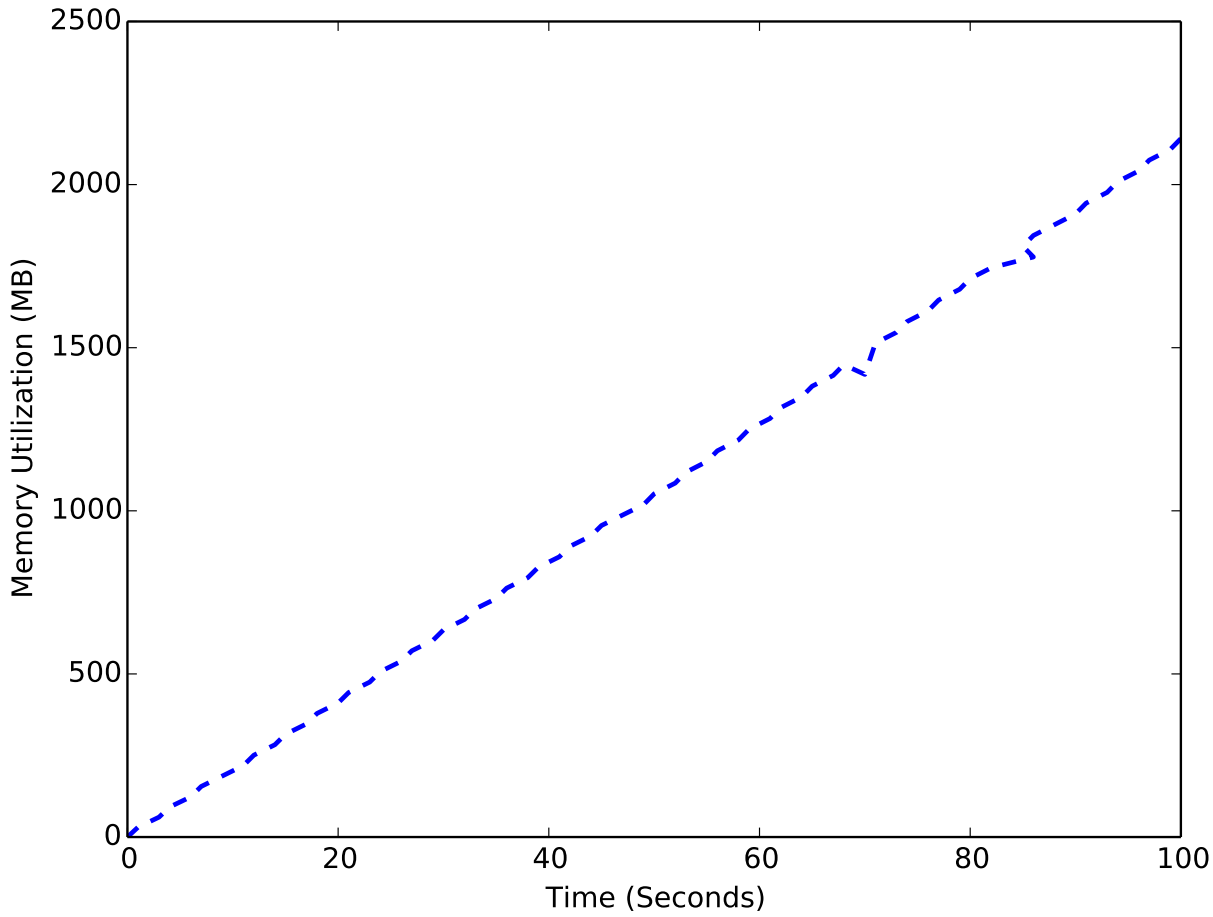

Figure 4.4: Provenance file size growth rate.

Chapter 5

IoT Device Sensor Data Anomaly Detection using Provenance Graphs

The Internet of Things (IoT) has revolutionized the way humans interact with devices. Unfortunately, this technological revolution has been met with privacy and security challenges. One major concern is the notion of malicious intrusions. A common way of detecting a malicious intrusion is by treating an attack as an anomaly and using anomaly detection techniques to detect the source of intrusion. In a given IoT device, provenance which denotes causality between system events offers immense benefit for intrusion detection. Provenance provides a comprehensive history of activity performed on a system's data, which indirectly ensures device trust. In this paper, we propose an approach to intrusion detection of IoT devices that leverages the observed behavior of sensor data flow as seen through provenance graphs. In this approach, an observed provenance graph is compared to an application's known provenance graph to determine if an anomaly exists. This comparison involves the dimensionality reduction of a provenance graph to a vector space representation and similarity metric. We evaluated our approach on an IoT application which simulates a climate control system.

5.1 Introduction

IoT devices have become an essential part of our daily lives in commercial, industrial, and infrastructure systems. These devices offer immense benefits to consumers by interacting with our physical environment through sensors and actuators which allows device automa-

tion thereby improving efficiency. Unfortunately, the proliferation of IoT devices has led to an increase in the number of remotely exploitable vulnerabilities leading to new attack vectors that could have disastrous financial and physical implications [6]. In 2015, security researchers demonstrated a vulnerability on the Jeep vehicle which allowed remote control of the automotive system over the Internet [21]. In 2016, researchers discovered a vulnerability that allows internet connected smart thermostats to be subject to remote ransomware attacks in which an attacker gains sole control of the thermostat until a fee is paid [49]. These are a few example of some of the potential malicious vulnerabilities that could have devastating long lasting impact on an IoT system.

Intrusion detection [31] is the process of discovering malicious exploits in a system. One way of detecting an intrusion is by the use of anomaly detection techniques. An anomaly, also referred to as an outlier, is data that deviates from the normal system behavior. Anomalies could be indicative of a system fault or that a system has been compromised by a malicious event. Due to the sensitive nature of safety critical systems, detecting malicious attacks is of utmost importance.

We propose an approach to identifying anomalous sensor events using provenance graphs. This approach involves the use of a similarity metric to compare observed provenance graphs with provenance graph derived from an application's normal execution. The result is an anomaly score which is compared with a previously set threshold to classify observed provenance graph as either anomalous or benign. We evaluate the effectiveness of our approach with a sample IoT application which simulates a climate control system.

5.2 BACKGROUND

5.2.1 Anomaly Detection

The notion of what constitutes an anomaly is often domain specific. Hawkins defines an anomaly as an "observation which deviates so much from other observations as to

arouse suspicion that it was generated by a different mechanism" [24]. In computing, an anomaly often indicates the presence of a malicious intrusion or a system fault. For example, an anomaly could be a sudden increase in web traffic on a web server which could be indicative of a denial of service attack. Additionally, in a critical care health device such as a pacemaker, an anomalous event could be detrimental to the health of a patient which could result in the loss of life.

Anomaly detection consists of two phase, learning phase also known as the training phase and the test also known as the detection phase. In the training phase, the system collects training dataset. This data is considered to be a representation of the system's normal daily activity and free from malicious events. Once training dataset has been collected, the system's activities are further observed. This part is known as the testing phase. In the testing phase, observed system behavior is compared to the learning phase to determine is an anomaly exists between the two. A threshold as defined by domain experts is used to determine if the observed data is considered an anomaly. Data labels are grouped into three major categories. Supervised anomaly detection, semi-supervised anomaly detection, and unsupervised anomaly detection. In supervised anomaly detection approach, training data contains instances of normal and anomalous data. Incoming data is classified based on the training data category. In semi-supervised approach, only one class of training data is collected, normal data. Incoming data that does not fit the normal class as specified by a threshold is regarded as an anomalous. In the training phase, most anomaly detection techniques use data derived from the normal system behavior. This is referred to as one class classification. In unsupervised approach, there are no training dataset. it is believed that normal data is clustered around each and occurs more frequently than an anomaly. This is a widely used form of anomaly detection since training data which is hard to get is not requited.

Anomaly detection involves the use of rule-based, statistical, clustering or classification techniques to determine normal or anomalous data instances. The process of determining all anomalous instances in a given dataset or system is a complex task. A major challenge

in anomaly detection is providing the right feature set from the data to use for detection. Another challenge exists in defining what constitutes as normal system behavior. Most anomaly detection using point based data often fail to include the dependencies that exist between data points.

Details on anomaly detection techniques are outlined below

1. Statistical-based approach: This approach involves the use of parametric or non parametric statistical inference to develop models which are used to determine if a dataset fits a statistical model. Instances that do not fit the defined statistical model are classified as an anomaly.

2. Classification: The main idea in this approach involves building models which use training data set with predefined labels (i.e normal, anomalous) to classify incoming data. Classification works in two phase: training phase and observation phase. In the training phase, data is collected which contains labels of normal and anomalous system behavior. If the dataset only contains a label of either anomalous or normal behavior, this is considered as a one class classifier. In the observation phase, incoming data is classified by defined data labels.

3. Nearest-Neighbor: It is based on the assumption that normal data occurs in dense neighborhoods and abnormal data in sparse neighborhoods. The main idea is to assign an incoming observation data to a class based on its proximity to the closest data point in the training data set. A distance or similarity measure is used to quantify the distance between points in a dataset. A popular form of nearest neighbor technique is the k-nearest neighbor which groups incoming data based on proximity to k closest data point. Details on k- nearest neighbors is discussed in section 3.5.

4. Clustering: The main idea is to group similar data instances into clusters. There are various approach to clustering. One approach looks at the density of the clusters, normal data belongs to large dense clusters while abnormal data belongs to small

clusters. Another approach as treats clustering as one class which assumes that normal belongs to a cluster and abnormal data does not. Another approach looks at the distance of the data to the centroid. Centroids are seen as the center of the cluster. Normal data are considered to be closer to the centroid than anomalies lie.

5. Density: This approach is used to estimate the density of k nearest neighbors. A data instance in a dense neighborhood is considered normal while data instances in neighborhoods with a sparse density are considered anomalous. The distance from a data instance to a nearest neighbor is seen as the inverse of the density of data instances. This approach faces an issue in which the density approach performs poorly in regions of varying densities. Local outlier factor addresses this issue. Local outlier factor is a measure of the degree of Outlieriness of each data instance contained in a data set. It is achieved by comparing the ratio of local density of k nearest neighbors to the density of a data instance. Data instances with lower density are considered outliers.

5.2.2 Provenance Graphs

Provenance denotes the origin of an object and all other activities that occurred on that object. An example of provenance can be seen with a college transcript. A transcript is the provenance of a college degree because it outlines all of the courses satisfied in order to attain a degree. In computing, data provenance, also known as data lineage, can be defined as the history of all activities performed on a data object from its creation to its current state. Provenance ensures data trust [11]. It establishes causality between entities contained in a data object through information flow tracking thereby it allows for the verification of a data source. Provenance is represented by a directed acyclic graph (DAG) in which vertices represent various data entities and the edges correspond to the interaction between them.

Most provenance collection frameworks developed to track provenance use system level

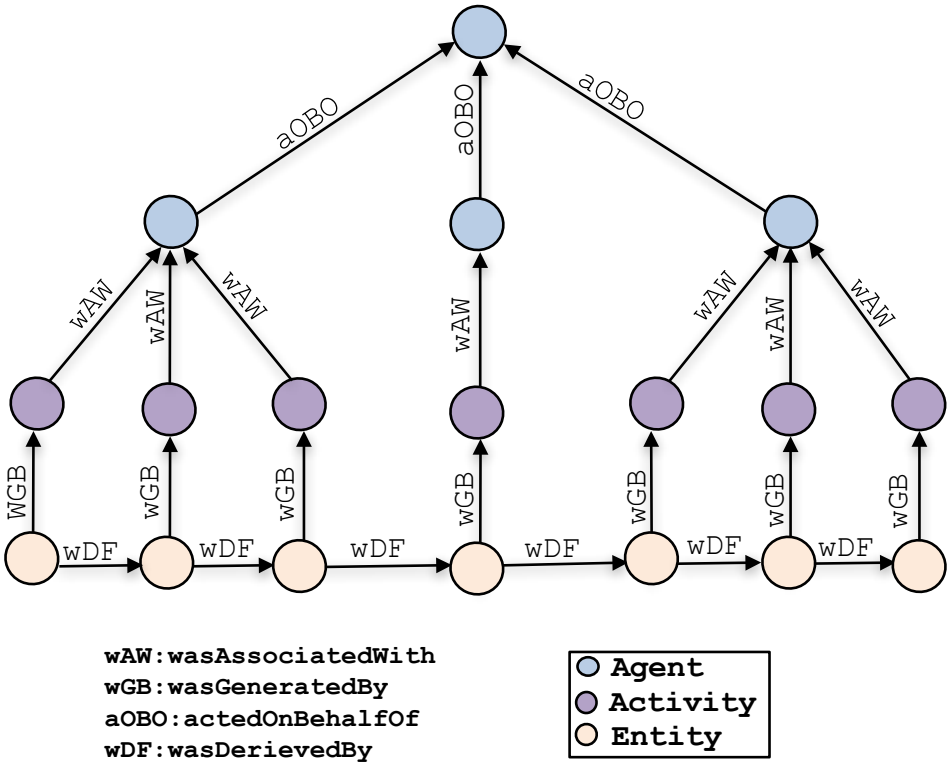

Figure 5.1: Components of a provenance graph where nodes represents types (agent, activity, and entity) and edges represents relationship between types

event sequences in an operating system [46, 57, 39]. For most IoT devices, containing limited or no operating system functionality, it is essential to use a provenance collection framework that places less emphasis on an operating system and more emphasis on application level information flow tracking. For our provenance graph generation, we use PAIoT [43], a provenance collection framework which tracks the information flow of sensor based events in an IoT device over its lifecycle. We formally define a provenance graph as follows:

Definition 3 *A provenance graph is a directed acyclic graph,* $p = (V, E)$ *where* V *is a set of vertices* $V = \{v_1, ..., v_n\}$ *such that* $v_i = (label, type, value)$*. Two vertices* v_x*,* v_y *are similar (denoted* $v_x \sim v_y$*) if* $v_x.type = v_y.type$ *and* $v_x.value = v_y.value$ *. Types may take on*

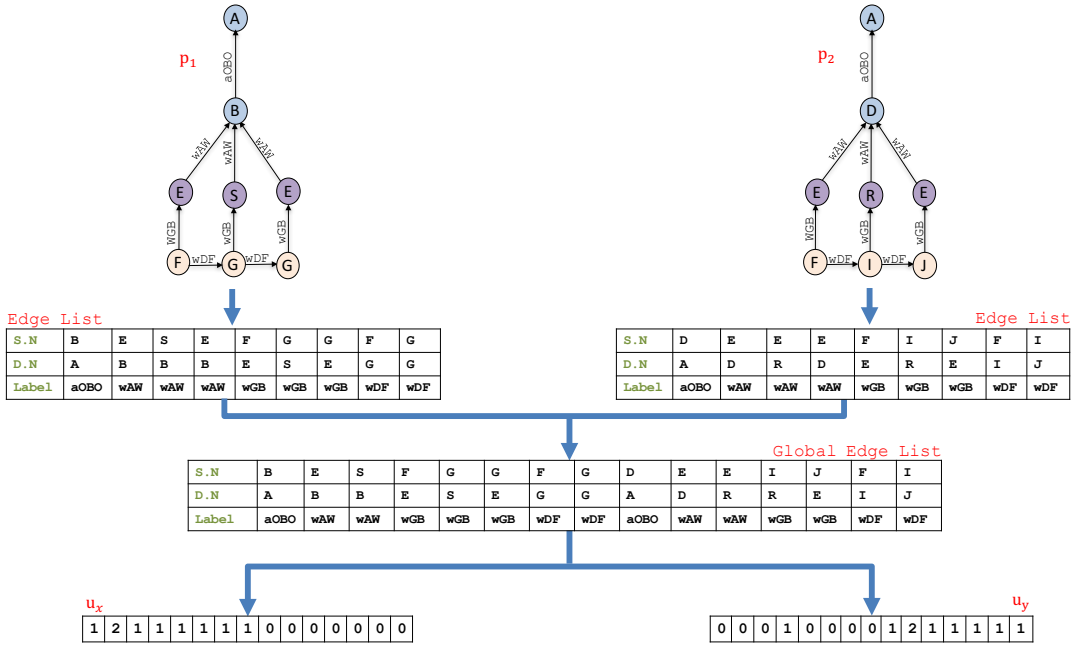

Figure 5.2: Provenance graph conversion to vector space. u_x, u_y represents the vectors generated from both provenance graphs

one of the following: Agent, Entity, and Activity. An agent is an actor that is represented by an identifier (e.g. sensor or device name). An entity is an event, which represents data that is produced as a result of an activity. An activity is an action performed by an agent or entity (e.g., read, update, create, delete). E is a set of edges $E = \{e_1, ..., e_n\}$ *where* $e_i = (v_s, v_d, label)$ *and* v_s, v_d *are source and destination vertices. label takes on one of the following values: wasGeneratedBy, used, wasInformedBy, wasDerivedFrom, wasAttributedTo, wasAssociatedWith, actedOnBehalfOf.*

Many IoT devices implement a control systems in which sensor data is used as an input in a feedback loop to an actuator. The operations of most control systems are regular and predictable. This notion can be leveraged to define an expected provenance graph for each application. Each iteration of a control loop sequence generates a path in a provenance graph. The nodes contained along the path denote historical event transformations. For

example, in a thermostat application, temperature readings generated might be converted from Celsius to Fahrenheit and utilized as feedback to an actuator. We focus on networked connected control system. Network connection opens doors to potential vulnerabilities

The expected regularity of provenance graphs in IoT applications motivate a graph-based approach to anomaly detection. This approach consists of two phase: observation phase, also known as the training phase and the test or detection phase. In the observation phase, the system collects provenance data considered to be a representation of the normal system behavior. In the detection phase, provenance graph from the observation phase is compared with the provenance graph derived from subsequent observations to determine if an anomaly exists.

5.2.3 Similarity Metric

Similarity metric is a measure of how identical two objects are, for example, by measuring the angle between objects (using cosine similarity) or a linear distance (using euclidean distance) between the objects. In this work, we use cosine similarity as our similarity metric. Cosine similarity is a measure of orientation between two non-zero vectors. It measures the cosine of the angle between the vectors. Two vectors which are at an angle of 90°have a similarity of 0, two vectors which are identical (with an angle of 0°) have a cosine of 1, and two vectors which are completely opposite (with an angle of 180°) have a similarity of -1. Since we are concerned with the similarity of the vectors, we are only concerned with the positive values bounded in [0,1]. To compute the cosine similarity between two vectors, X and Y:

$$\cos(\theta) = \frac{X \cdot Y}{\|\mathbf{X}\| \cdot \|\mathbf{Y}\|} = \frac{\sum_i^n X_i Y_i}{\sqrt{\sum_i^n X_i^2}\sqrt{\sum_i^n Y_i^2}}$$

Jaccard Similarity

This similarity measure evaluates the intersection divided by the union of two non zero vectors.

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

Euclidean distance

This measures calculates the line distance between two data objects in an euclidean space. The euclidean distance between vectors X and Y , $d(X,Y)$ is defined by:

$$d(X,Y) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2}$$

Our method of comparing the similarity of provenance graphs was inspired by an information retrieval technique for document retrieval. Given a corpus $D = \{d_1, ..., d_n\}$, and query, q, how do we find document(s) $\{d_x, ...d_y\}$ which are similar to q and rank them by order of importance. To achieve this, documents contained in the corpus are converted into a vector space representation which allows documents to be ranked based on some similarity metric.

5.3 Threat Model and Assumptions

Due to the ubiquitous nature of IoT devices, there are a wide array of vulnerabilities associated with them. In designing our anomaly detection framework, we expect an attacker's footprint is reflected through the data flow as depicted in the provenance graph. Our algorithm detects attacks such as false data injection, and state change as depicted in information flow of sensor events in provenance graphs.

5.4 Graph Similarity Anomaly Detection

As discussed in section 5.2.3,a similarity function measures how identical two provenance graphs are in a vector space. In order to apply a similarity function to a provenance graph, we need to compute a vector representation. In our approach, a provenance graph undergoes dimensionality reduction into an n-dimensional vector space where n represents the number of unique edges (global set of edges) contained in the provenance graph set. This representation is used as input to our anomaly detection algorithm.

Selecting the most appropriate feature from provenance graphs is an important task. We need to utilize features that preserve the order of edges and nodes contained in the provenance graph. Our approach utilizes the occurrence of unique edges contained in both graphs.

We provide a means of classifying such edges as identical. To determine identical edges, we utilize a combination of edge labels and source and destination node. That is, two edges e_x and e_y are similar (denoted $e_x \sim e_y$) if $e_x.v_s \sim e_y.v_s$, $e_x.v_d \sim e_y.v_d$, and $e_x.label = e_y.label$

An edge list E_p consists of all edges contained in p. A global edge list E_G consists of all non-similar edges contained in P. E_G preserves the ordering of edges contained in P.

Definition 4 *Let* $P = \{p_1, ..., p_n\}$ *such that* $p_i = (V, E)$. $E_G = \{e_1, ..., e_n\} \mid E_G \in P$ *where* $e_i \neq e_{i+1}, 1 \leq i \leq n$. $E_p \leftarrow \{e_1, ..., e_n\} \mid E_p \in p$ *where* $e_{pi} \neq e_{pi+1}, 1 \leq i \leq n$

Finally, we define our provenance graph to vector approach as follows:

Definition 5 *Let* $P = \{p_1, ..., p_n\}$ *where* $p_i = (V, E)$. *The vector space representation of* $p_i, \boldsymbol{u_i}$, *is the number of times edges contained in* E_p *occurs in* E_G. $\boldsymbol{u_i}$ *has a length of* k *which consists of all of the unique edges contained in the global edge list* E_G.

Figure 5.2 displays the vector space conversion of provenance graphs. $\boldsymbol{p_1}$, and $\boldsymbol{p_2}$ consists of vertices $A, B, E, F, G, I, J, K, M$ and edges $aOBO, wAW, wGB, wDF$. The vector representation of graphs, u_1, and u_2 is the occurrence of edges contained in the edge list that is found in the global edge list.

5.4.1 Graph Based Similarity Detection Algorithm

Given u_x, u_y which denotes the vector representation of provenance graphs p_x, p_y. The similarity of p_x, p_y is found by calculating the cosine similarity between the two vectors where 1 denotes similarity between the two vectors and 0 denotes non-similarity between the two vectors. A threshold value is set which is used to classify the behavior of provenance graphs in the detection phase as normal or an anomaly.

$$sim(u_x, u_y) = \frac{u_x \cdot u_y}{\|\mathbf{u_x}\| \cdot \|\mathbf{u_y}\|} = \frac{\sum_{i=1}^{n} u_{xi} u_{yi}}{\sqrt{\sum_{i=1}^{n} u_{xi}^2}\sqrt{\sum_{i=1}^{n} u_{yi}^2}} \in [0, 1]$$

Algorithm 1 depicts a formal definition of our graph-based anomaly detection algorithm.

Defining Anomaly Threshold

An anomaly threshold t is a score that defines at what point a provenance graph contained in the test data is considered anomalous. Ensuring a proper threshold score is used for detection is an important task that requires extensive knowledge of the application domain. The threshold is manually set to a value t, which is defined by domain experts. For automatic anomaly threshold detection, one can use prediction methods to define an anomaly score. Prediction techniques are beyond the scope of this research.

5.5 Experiment

This section outlines experimental procedures involved in evaluating our proposed algorithm.

5.5.1 Evaluation

The experiment evaluation serves as preliminary study to confirm the correctness of our theoretical approach in detecting anomalous instances between provenance graphs. We

Algorithm 2 Graph Set to Edge Conversion

procedure GRAPHSETTOEDGELIST(P)
  INPUT: $P = \{p_0, ..., p_n\} \mid p_i \leftarrow (V_i, E_i), 0 \leq i < n.$
  $E_G \leftarrow \{\}$
  for $p_i = (V_i, E_i) \in P, 0 \leq i < n$ **do**
    for $e_j \in E_i$ **do**
      Found $\leftarrow$ False
      for $e_g \in E_G$ **do**
        if $e_j \sim e_g$ **then**
          Found $\leftarrow$ True
        end if
      end for
      if *Found* = False **then**
        $E_G \leftarrow E_G \cup e_j$
      end if
    end for
  end for
  return E_G
end procedure

evaluate our intrusion detection algorithm by implementing an IoT application which simulates a climate control system. Constant irregularities in temperature could have devastating effects on industrial machinery. Climate control systems ensures a proper functional environment for people and machinery. This system consist primarily of a Heating Ventilating and Air Conditioning (HVAC) System which uses temperature and humidity data to regulate environmental conditions within a building. The temperature is set within a bounded limit such that when the temperature exceeds a certain threshold, a cooling phase is activated and the air conditioning is switched on until the temperature decreases below

Algorithm 3 Graph to Vector Conversion

procedure GRAPHTOVECTOR(E, E_G)
  $k = |E_G|$
  $Q[k] \mid Q[i] \leftarrow 0, 0 \leq i < k$
  for $e_j \in E$ **do**
    for $e_g \in E_G \mid 0 \leq g < k$ **do**
      if $e_j \sim e_g$ **then**
        $Q[g] \leftarrow Q[g] + 1$
        $Found \leftarrow$ **True**
      end if
    end for
  end for
  return Q
end procedure

Algorithm 4 Graph Anomaly Detection

procedure GRAPHANOMALY(P, p)
  $E_G \leftarrow GraphSetToEdgeList(P \cup p)$
  $Q \leftarrow GraphtoVector(p, E_G)$
  for $p_i \in P$ **do**
    $N_i \leftarrow GraphtoVector(p_i, E_G)$
    $z \leftarrow sim(Q, N_i)$
    if z $\geq$ threshold **then**
      return normal
    end if
  end for
  return anomaly
end procedure

a threshold in which the heating phase is activated in which the heater is turned on. We utilize a publicly available dataset [30] which consists of a year's worth of longitudinal data on the thermal conditions, related behaviors, and comfort of twenty-four occupants in a medium-sized building. This dataset consist of temperature, humidity, air velocity generated at a duration of fifteen minutes. We utilize the temperature and humidity data as input to our aforementioned simulation program. We generate provenance graphs for each week of the year. We compare the provenance graph generated in various weeks to see how they differ using our graph similarity approach (e.g week 1 compared to week 2, week 2 compared to week 3).

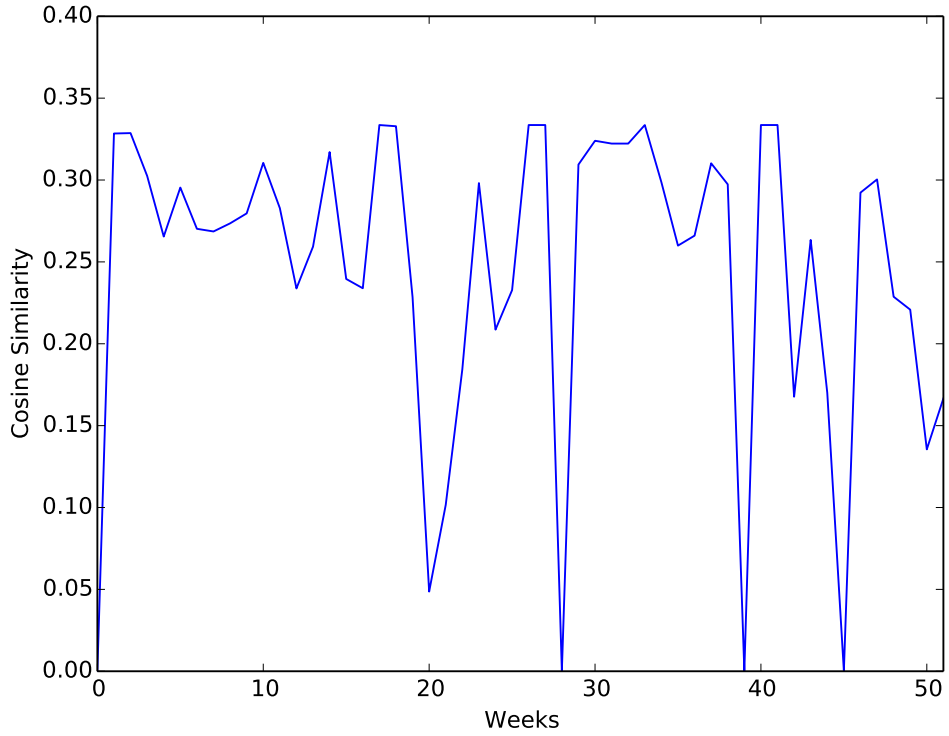

Figure 5.3: Provenance graph comparison of climate control system by week

Figure 5.3 depicts the cosine similarity between provenance graphs generated from the first occupant by preceding weeks. It is worth nothing that the rapid decline in the graph is as a result of anomalies between the provenance graphs. This might be as a result of

rapid variation in temperature change for each week in the given dataset.

5.6 Related Work

There has been a considerable amount of research done on anomaly detection most of which involves the analysis of system call sequence. Liao et al [33] characterizes a system's normal behavior by denoting the frequency of unique system calls which are converted into a vector space representation. A classification algorithm such as k-Nearest Neighbors was used to classify the training data set. Stephanie et al [25] defines a link between the human immune system and intrusion detection systems. They developed a normal system behavior repository by analyzing system call sequences of an application. The application's system call sequence is stored in a normal database which is queried during observation. If an application executes a sequence of system calls that is not found in the normal database, a mismatch is recorded. If the mismatch for that application exceeds a threshold, an anomaly is detected. Yoon et al. [56] developed a technique for intrusion detection on embedded systems by analyzing system call frequencies. This is achieved by learning normal system profile of observed patterns in the system call frequency distribution.Their hypothesis is that applications follow a known frequency pattern which is centered around the centroid. Data from the training set is clustered using k-means which groups legitimate system behavior. Observation at run-time are compared with the clusters in the detection phase, if the incoming observation does not fit into a cluster, it is considered an anomaly. Additionally, anomaly detection on graphs has also been explored. Manzoor et al [36] proposed a centroid based clustering anomaly detection for instances of streaming heterogeneous graphs in real time. Papadimitriou et al [44] proposed five similarity algorithms for comparing the similarity of web graphs namely signature similarity, vertex/edge vector similarity, vertex ranking,and vertex edge overlap.

[42] proposed two algorithms for anomaly detection in graphs The first approach, looks at unusual substructures in graphs. This is achieved by inverting the subdue score of

patterns that occurs frequently in a graph. Subdue is a method for detecting substructures within graphs. The values that produce high scores are flagged as anomalies. The second approach examines unusual subgraphs contained in a graph by iteratively using subdue algorithm to compress the graph. The idea behind this method is that subgraphs containing many common substructures are considered less anomalous than one with less substructures.

Some graph approach involve the use of a community-based approach in which dense regions of connected nodes are considered normal and nodes with high sparse regions which do not belong to any community are considered anomalous. *AUTOPART* [14] consist of nodes with similar neighbors are clustered together and the edges which do not belong to any cluster are considered as an anomaly. To detect communities, it achieves this task by reorganizing the rows and the columns of the adjacency list.

Our approach of graphs similarity is similar to graph kernels and graph edit distance. graph kernels involves measuring the similarity between two graphs, graph edit distance looks at the number of operations required for a graph G_1 to be identical to G_2.

5.7 Summary and Conclusion

In this paper, we propose an anomaly detection algorithm for detecting anomalous instances of sensor based events in an IoT device using provenance graphs. We evaluate our approach with a preliminary study on an IoT application which simulates a climate control system. In the future, we plan on conducting further experimentation to identify the false and true positive rates of our algorithm using select IoT application dataset.

Chapter 6

Conclusion

6.1 Summary

In this proposal, we motivate the need for integrating provenance into the IoT architecture. We propose a provenance collection framework that provides provenance collection capabilities for devices in the IoT. To address the scalability challenge of provenance storage, we also propose a policy-based extension to our framework that provides efficient provenance data storage.

6.2 Future Work

Some of the proposed future work for this research is:

- Provenance collection raises privacy issues. How do we ensure that the vast amount of data collected is not invasive to privacy.
- Provenance Versioning: Provenance version creates cycles. When a sensor-actuator data is read or edited, is a new instance of the file created? Tracking all transformations that occurs on a data object in memory constrained devices might lead to running out of storage space if each transformation creates a new copy of the object.
- Securing Provenance: Proper encryption and authentication techniques [23] are needed to ensure the confidentiality, and integrity of provenance data.

References

[1] PROV-DM: The PROV Data Model. https://www.w3.org/TR/prov-dm/. Accessed: 2016-10-01.

[2] PROV-JSON: The PROV Data Model. http://www.w3.org/Submission/2013/SUBM-prov-json-20130424, Apr 2013. Accessed: 2016-10-01.

[3] Micah Adler and Michael Mitzenmacher. Towards compressing web graphs. In *Proceedings of the Data Compression Conference*, DCC '01, pages 203–, Washington, DC, USA, 2001. IEEE Computer Society.

[4] Mufajjul Ali and Luc Moreau. A provenance-aware policy language (cprovl) and a data traceability model (cprov) for the cloud. In *Proceedings of the 2013 International Conference on Cloud and Green Computing*, CGC '13, pages 479–486, Washington, DC, USA, 2013. IEEE Computer Society.

[5] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In Luc Moreau and Ian Foster, editors, *Provenance and Annotation of Data*, number 4145 in Lecture Notes in Computer Science, pages 118–132. Springer Berlin Heidelberg, May 2006. DOI: 10.1007/11890850_14.

[6] Mario Ballano Barcena and Candid Wueest. Insecurity in the internet of things.

[7] Adam Bates, Kevin R B Butler, and Thomas Moyer. Linux provenance modules: Secure provenance collection for the linux kernel. 2014.

[8] Adam Bates, Kevin R. B. Butler, and Thomas Moyer. Take only what you need: Leveraging mandatory access control policy to reduce provenance storage costs. In

Proceedings of the 7th USENIX Conference on Theory and Practice of Provenance, TaPP'15, pages 7–7, Berkeley, CA, USA, 2015. USENIX Association.

[9] Adam Bates, Ben Mood, Masoud Valafar, and Kevin Butler. Towards Secure Provenance-based Access Control in Cloud Environments. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 277–284, New York, NY, USA, 2013. ACM.

[10] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. Trustworthy whole-system provenance for the linux kernel. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 319–334, 2015.

[11] Elisa Bertino. *Data Trustworthiness—Approaches and Research Challenges*, pages 17–25. Springer International Publishing, Cham, 2015.

[12] Uri Braun, Simson Garfinkel, David A Holland, Kiran-Kumar Muniswamy-Reddy, and Margo I Seltzer. Issues in automatic provenance collection. In *International Provenance and Annotation Workshop*, pages 171–183. Springer, 2006.

[13] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and Where: A Characterization of Data Provenance. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory ICDT 2001*, number 1973 in Lecture Notes in Computer Science, pages 316–330. Springer Berlin Heidelberg, January 2001. DOI: 10.1007/3-540-44503-X_20.

[14] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 112–124. Springer, 2004.

[15] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in Databases: Why, How, and Where. *Found. Trends databases*, 1(4):379–474, April 2009.

[16] Dave Evans. The internet of things how the next evolution of the internet is changing everything. `https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`, Apr 2011. Accessed: 2016-10-01.

[17] Ian T. Foster, Jens-S. Vöckler, Michael Wilde, and Yong Zhao. Chimera: Avirtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, SSDBM '02, pages 37–46, Washington, DC, USA, 2002. IEEE Computer Society.

[18] Eleni Gessiou, Vasilis Pappas, Elias Athanasopoulos, Angelos D. Keromytis, and Sotiris Ioannidis. Towards a Universal Data Provenance Framework Using Dynamic Instrumentation. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research*, number 376 in IFIP Advances in Information and Communication Technology, pages 103–114. Springer Berlin Heidelberg, June 2012. DOI: 10.1007/978-3-642-30436-1_9.

[19] Devarshi Ghoshal and Beth Plale. Provenance from Log Files: A BigData Problem. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 290–297, New York, NY, USA, 2013. ACM.

[20] Boris Glavic, Kyumars Sheykh Esmaili, Peter M. Fischer, and Nesime Tatbul. The Case for Fine-Grained Stream Provenance. In *Proceedings of the 1st Workshop on Data Streams and Event Processing (DSEP) collocated with BTW*. 2011.

[21] Andy Greenberg. The jeep hackers are back to prove car hacking can get much worse. 2015.

[22] Paul Groth, Simon Miles, and Luc Moreau. PReServ: Provenance recording for services. In *In, UK e-Science All Hands Meeting 2005, Nottingham, UK, EPSRC*.

[23] Ragib Hasan, Radu Sion, and Marianne Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *Proccedings of the 7th Conference*

on File and Storage Technologies, FAST '09, pages 1–14, Berkeley, CA, USA, 2009. USENIX Association.

[24] D. M. Hawkins. *Identification of outliers*. Monographs on applied probability and statistics. Chapman and Hall, London [u.a.], 1980.

[25] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *J. Comput. Secur.*, 6(3):151–180, August 1998.

[26] Syed Rafiul Hussain, Changda Wang, Salmin Sultana, and Elisa Bertino. Secure Data Provenance Compression Using Arithmetic Coding in Wireless Sensor Networks. *2014 IEEE International Performance Computing and Communications Conference (IPCCC)*, December 2014.

[27] Yang Ji, Sangho Lee, and Wenke Lee. RecProv: Towards Provenance-Aware User Space Record and Replay. In Marta Mattoso and Boris Glavic, editors, *Provenance and Annotation of Data and Processes*, number 9672 in Lecture Notes in Computer Science, pages 3–15. Springer International Publishing, June 2016. DOI: 10.1007/978-3-319-40593-3_1.

[28] David Reinsel John Gantz. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. `https://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf`, apr 2012. Accessed: 2016-10-01.

[29] Samuel T. King and Peter M. Chen. Backtracking intrusions. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 223–236, New York, NY, USA, 2003. ACM.

[30] Jared Langevin, Patrick L. Gurian, and Jin Wen. Tracking the human-building interaction: A longitudinal field study of occupant behavior in air-conditioned offices. *Journal of Environmental Psychology*, 42(Supplement C):94 – 115, 2015.

[31] Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. *Intrusion Detection: A Survey*, pages 19–78. Springer US, Boston, MA, 2005.

[32] Kristen LeFevre and Evimaria Terzi. *GraSS: Graph Structure Summarization*, pages 454–465.

[33] Yihua Liao and V. Rao Vemuri. Using Text Categorization Techniques for Intrusion Detection. In *Proceedings of the 11th USENIX Security Symposium*, pages 51–59, Berkeley, CA, USA, 2002. USENIX Association.

[34] Hyo-Sang Lim, Yang-Sae Moon, and Elisa Bertino. Provenance-based Trustworthiness Assessment in Sensor Networks. In *Proceedings of the Seventh International Workshop on Data Management for Sensor Networks*, DMSN '10, pages 2–7, New York, NY, USA, 2010. ACM.

[35] Peter Macko and Margo Seltzer. A general-purpose provenance library. In *Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance*, TaPP'12, pages 6–6, Berkeley, CA, USA, 2012. USENIX Association.

[36] Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1035–1044, New York, NY, USA, 2016. ACM.

[37] Friedemann Mattern and Christian Floerkemeier. From Active Data Management to Event-based Systems and More. pages 242–259. Springer-Verlag, Berlin, Heidelberg, 2010.

[38] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware Storage Systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[39] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[40] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo Seltzer. Provenance for the Cloud. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, pages 15–14, Berkeley, CA, USA, 2010. USENIX Association.

[41] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 419–432, New York, NY, USA, 2008. ACM.

[42] Caleb C. Noble and Diane J. Cook. Graph-based anomaly detection. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 631–636, New York, NY, USA, 2003. ACM.

[43] Ebelechukwu Nwafor, David Hill, Andre Campbell, and Gedare Bloom. Towards a provenance aware framework for internet of things devices. In *Proceedings of the 14th International Conference on Ubiquitous Intelligence and Computing*, UIC '17, San Fransisco, CA, USA, 2017. IEEE Computer Society.

[44] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, May 2010.

[45] J. Park, D. Nguyen, and R. Sandhu. A provenance-based access control model. In *2012 Tenth Annual International Conference on Privacy, Security and Trust (PST)*, pages 137–144, July 2012.

[46] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In *Symposium on Cloud Computing (SoCC'17)*. ACM, ACM, 2017.

[47] Devin J. Pohly, Stephen McLaughlin, Patrick McDaniel, and Kevin Butler. Hi-Fi: Collecting High-fidelity Whole-system Provenance. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 259–268, New York, NY, USA, 2012. ACM.

[48] Rationality. *The Cambridge Dictionary of Philosophy*. Cambridge University Press, 1999.

[49] Dan Raywood. Defcon: Thermostat control hacked to host ransomware. 2016.

[50] Koen Smets and Jilles Vreeken. The odd one out: Identifying and characterising anomalies. In *SDM*, 2011.

[51] R. Spillane, R. Sears, C. Yalamanchili, S. Gaikwad, M. Chinni, and E. Zadok. Story Book: An Efficient Extensible Provenance Framework. In *First Workshop on the Theory and Practice of Provenance*, TAPP'09, pages 11:1–11:10, Berkeley, CA, USA, 2009. USENIX Association.

[52] Mark Stanislav. Hacking iot: A case study on baby monitor exposures and vulnerabilities. `https://www.rapid7.com/docs/Hacking-IoT-A-Case-Study-on-Baby-Monitor-Exposures-and-Vulnerabilities.pdf`, Sep 2015. Accessed: 2016-10-01.

[53] Dawood Tariq, Maisem Ali, and Ashish Gehani. Towards Automated Collection of Application-level Data Provenance. In *Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance*, TaPP'12, pages 16–16, Berkeley, CA, USA, 2012. USENIX Association.

[54] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 567–580, New York, NY, USA, 2008. ACM.

[55] Hayawardh Vijayakumar, Joshua Schiffman, and Trent Jaeger. Integrity walls: Finding attack surfaces from mandatory access control policies. In *7th ACM Symposium on Information, Computer, and Communications Security (ASIACCS)*, May 2012.

[56] Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Mihai Christodorescu, and Lui Sha. Learning execution contexts from system call distribution for anomaly detection in smart embedded system. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, pages 191–196, New York, NY, USA, 2017. ACM.

[57] Robert H'obbes' Zakon, editor. *28th Annual Computer Security Applications Conference, ACSAC 2012, Orlando, FL, USA, 3-7 December 2012*. ACM, 2012.

[58] Shams Zawoad and Ragib Hasan. Fecloud: A trustworthy forensics-enabled cloud architecture. In *11th Annual IFIP WG 11 International Conference on Digital Forensics*, 2015.