DATA PROVENANCE FOR INTERNET OF THINGS (IOT)

EBELECHUKWU NWAFOR

Department of Electrical and Computer Science

APPROVED:

Gedare Bloom, Ph.D.

Legand Burge, Ph.D.

Wayne Patterson, Ph.D.

Charles Kim, Ph.D.

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

DATA PROVENANCE FOR INTERNET OF THINGS (IOT)

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 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 to make 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 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.

In this dissertation proposal, we explore the integration of provenance within the IoT. We take an in-depth look at the collection and use of provenance data in mitigating malicious attacks. We propose a provenance collection framework for IoT applications. We focus on a holistic approach in which provenance information is distributed throughout the IoT. We overcome the storage challenge of provenance collection by adopting a policy approach for provenance collection and storage which allows flexibility in deciding what provenance data to keep. 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 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 [14], 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 [34], 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 occured in order to provide preventive measures to eradicate future or current attacks. [13].

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 [17]. 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) [24, 6, 17]. 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(ie unrelated provenance data which is not required by an organization). By doing this, we provide pruning capabilities [11] 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 1.1 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.

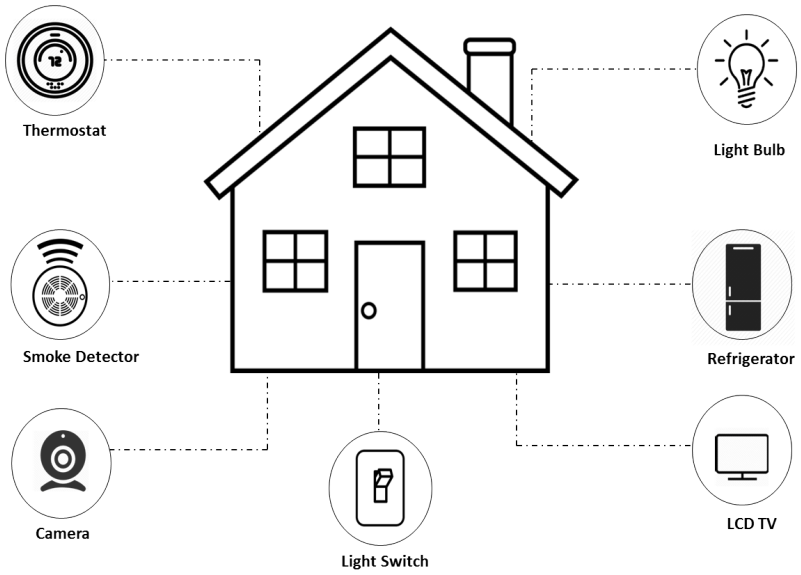

Figure 1.1: Smart home use case Diagram

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:

- 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.
- Modeling of 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.
- How do we effectively collect provenance data in resource constrained devices and relate this information across layers of the IoT architecture

1.4 Research Contribution

In this dissertation, we propose to make the following key contributions:

- A provenance collection framework which represents causality and dependencies between entities contained in an IoT system. This system creates groundwork for capturing and storing provenance data across the IoT architectural stack.
- A policy driven approach for efficient provenance data storage on IoT devices. We evaluate our contributions for an efficient provenance storage system by using an intrusion detection system on IoT devices.

1.5 Organization of Dissertation proposal

This dissertation proposal is organized as follows. In Chapter 2, we present background information on data provenance and the Internet of Things. We also discuss models for representing provenance data and an overview of relevant compression techniques. In Chapter 3, we outline the state of the art on provenance collection. In Chapter 4, we discuss our proposed provenance collection system. We also describe our model for representing provenance data across the IoT architecture. In Chapter 5, we describe our framework for efficient storage of provenance data. Finally, in Chapter 6, we conclude the proposal with proposed future work.

Chapter 2

Background Information

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 [32] 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.

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. [13] decribes provenance as the origin and history of data from its lifecycle. Buneman et al [12] 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 [17] 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 [18, 6] 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 [8, 29, 28] . Provenance denotes the who, where and why of data [13]. 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. Figure 1 illustrates an example from our use case of provenance above. It outlines a graphical representation of relationships contained in a server's log file.

Provenance ensures trust and integrity of data [10]. 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 [38] to investigate the cause of a malicious attack and also in intrusion detection systems to further enhance the security of computing devices.

2.1.1 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.

2.2 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 [26] which represents a way in which the physical objects, "things", can be connected to the digital world. Gubbi et al [30] 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 with each other.*

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 [23] 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.

2.2.1 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 layers allows a better understanding on how to model the flow of information across actors contained in an IoT hierarchy.

Figure 2.1 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.

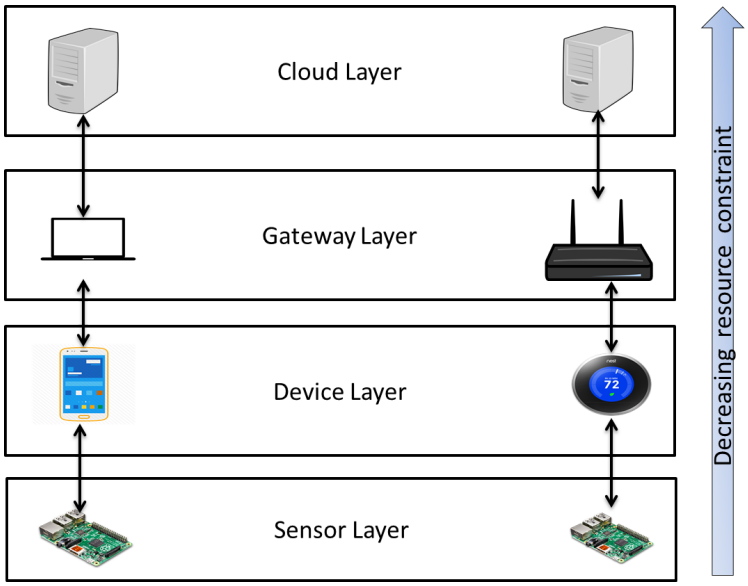

Figure 2.1: IoT Architecture Diagram. The arrows illustrates the interaction between data at various layers on the architecture.

2.3 Comparison of 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 Model for representing provenance for IoT

In order to represent the right kind of provenance information, we need to satisfy the who, where, how, and what of data transformations. Provenance data can be represented using a provenance model in a modeling language such as, PROVDM which is represented in serialized form as a JSON object. This model displays the causal relationship of data objects. We propose a model that contains information such as sensor readings, device name, and device information. There are two widely accepted modeling languages for representing provenance, PROV-DM [2] and Open Provenance Model [27] that have been applied in various literature and are considered standard for representing provenance. Details on the provenance models are outlined below.

2.4.1 Open Provenance Model (OPM)

Open Provenance Model [27] is a specification that was derived as a result of a meeting at the International Provenance and Annotation Workshop (IPAW) in May 2006. OPM was created to address the need of allowing a centralized model for representing provenance data amongst various applications. It allows for interoperability between various provenance

applications. The goal of OPM is to represent causality and dependency and also to develop a digital representation of provenance data regardless of if it is produced by a computer system. OPM is represented as a directed acyclic graph which denotes causal dependency between objects. The edges in the graph denotes dependencies with its source denoting effect and its destination denoting cause. The definitions of the edges and their relationships are denoted below:

- wasGeneratedBy: This edge denotes a relationship in which an entity(e,g artifact) is utilized by one or more entities(e.g process). An entity can use multiple entities so it is important to define the role.
- wasControlledBy: This edge outlines a relationship in which an entity caused the creation of another entity.
- used(Role): This edge denotes that an entity requires the services of another entity in order to execute.
- wasTriggeredBy: This edge denotes a relationship represents a process that was triggered by another process
- wasDerrivedFrom: This edge denotes a relationship which indicates that the source needs to have been generated before the destination is generated.

Objects represents the nodes of the relationship graph. There are three object contained in the OPM model: artifact, process, agent.

- artifact: An artifact represents the state of an entity. An artifact is graphically represented by a circle.
- Process: A process is an event which takes place at a particular time due to an agent or an artifact. A process is represented by a square object.

- Agent: Agents represents actors that facilitate the execution of a process. An agent is represented by a hexagon in an OPM graph.

OPM is used to represent all previous and current actions that have been performed on an entity and the relationship between each entities contained in the graph. Figure 2 represents an example of an OPM acyclic graph with all of its causal dependencies.

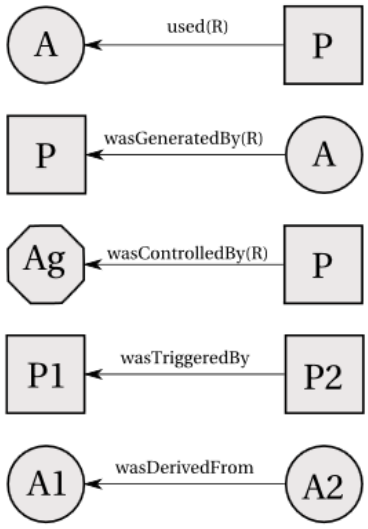

Figure 2.2: Edges and entities represented in OPM ©[27]

2.4.2 Provenance Data Model (Prov-DM)

Another model for representing provenance data is the Provenance Data Model (PROV-DM). PROV-DM is a W3C standardized extension of OPM. Prov-DM is a model that is used to depict causal relationships between entities, activities and agents (digital or physical). It creates a common model that allows for interchange of provenance information between heterogeneous devices. It contains two major components: types and relations.

- Entity: An entity is a physical or digital object. An example of an entity is a file system, a process, or an motor vehicle. An entity may be physical or abstract.

- Activity: An activity represents some form of action that occurs over a time frame. Actions are acted upon by an agent. An example of an activity is a process opening a file directory, Accessing a remote server.

- Agent: An agent is a thing that takes ownership of an entity, or performs an activity. An example of an agent is a person, a software product, or a process.

The figure below illustrates the various types contained in PROV-DM and their representation. Entities, activities and agents are represented by oval, rectangle and hexagonal shapes respectively.

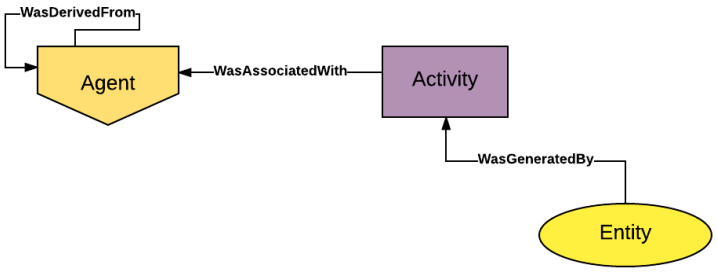

Figure 2.3: Prov-DM respresentation

Similar to the OPM, PROV-DM does not keep track of future events. PROV-DM relations are outlined below:

- wasGeneratedBy: This relation signifies the creation of an entity by an activity.

- used: This relation denotes that the functionality of an entity has been adopted by an activity.

- wasInformedBy: This relation denotes a causality that follows the exchange of two activities.

- wasDerivedFrom: This relation represents a copy of information from an entity.

- wasAttributedTo: This denotes relational dependency to an agent. It is used to denote relationship between entity and agent when the activity that created the agent is unknown.

- wasAssociatedWith: This relation denotes a direct association to an agent for an activity that occurs.This indicates that an agent plays a role in the creation or modification of the activity.

- actedOnBehalfOf: This relation denotes assigning authority to perform a particular responsibility to an agent. This could be by itself or to another agent.

Some of the difference between OPM and PROV-DM are described below:

- The main components Artifact, Process and Agent in the OPM model are modified to Entity, Action, and Agent.

- Additional causal dependencies such as wasAttributedTo and actedOnBehalfOf were included to represent direct and indirect causal dependencies respectively between agents and entities.

2.4.3 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 depicts the relationship in which s1 tries to access a file (FileB) which was generated by s2. PROV-JSON contains an entity, activity and agent which is represented as a json object and is identified by their respective ids. identifiers are optional in PROV-DM but are required for PROV-JSON since

json objects contains a 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 which 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:cityName": {
                "$": "Londres",
                "lang": "fr"
            }
            ...
        }
    },
    "agent": {
        "e1": {
            "ex:employee": "1234",
            "ex:name": "Alice",
            "prov:type": {
                "$": "prov:Person",
                "type": "xsd:QName"
            }
        }
        ...
    },

    "activity": {
        "a1": {
            "prov:startTime": "2011-11-16T16:05:00",
            "prov:endTime": "2011-11-16T16:06:00",
            "ex:host": "server.example.org",
            "prov:type": {
                "$": "ex:edit",
                "type": "xsd:QName"
            }
        }
    }

}
```

Figure 2.4: PROV-JSON MODEL ©[4]

2.5 P

olicy Model

2.5.1 eXtensible Access Control Markup Language

XACML is an access control policy language that allows the creation and enforcement of policies written in XML. It is a standard specification developed by the Organization of Advancement of Structured Information Standards(OASIS). Since it is written in XML, this framework allows for extensible and flexible policy documents. XACML consists of three major components which are involved in policy generation, evaluation and enforcement. The components of XACML are described below:

- Policy Administration Point (PAP): This component of XACML is involved with the generation of policy documents. A Policy document is created by specifications and requirements set aside by an administrator.
- Policy Decision Point (PDP): The Policy Decision Point evaluates policies by the request context generated by a user. Based on the request, It generates a response(accept,deny or intermediate) This response is communicated with the Policy Enforcement Point which enforces the response sent by the PDP.
- Policy Enforcement Point (PEP): The Policy Enforcement Point generates request context which is sent to the PDP for evaluation. It is also involved with the responsibility of enforcing request based on decision received by the PDP.

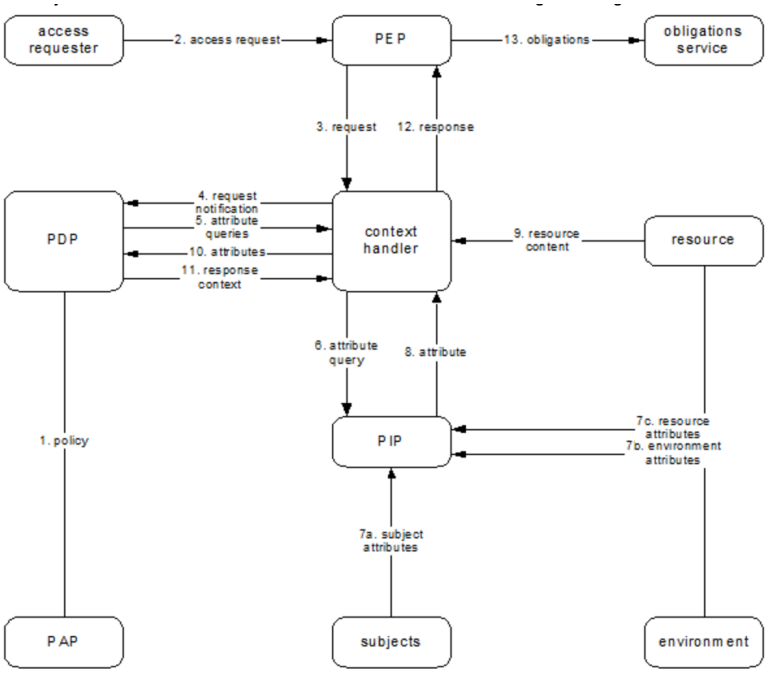

Figure 2.5: Xacml Data-flow diagram ©[1]

Chapter 3

Related Work

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 [3, 9, 15, 29]. 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 [28] 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 requirement since we are dealing with devices with limited memory and storage capabilities. We employ a policy based approach which allows to provenance pruning by storing what provenance data is considered important to a specific organization. Also,on like PASS, our system collects not just system level provenance but also application level provenance.

3.1.2 HiFi

Bates et al. [31] 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 primarily involve sensor and actuator readings.

On the other hand, HiFi deals with collecting system level events with might incurr additional overhead as 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 [22] 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 [33] 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 [8] 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. [35] 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 apporach for efficient provenance data storage. However, one major limitation exists in which provenance is only collected from function calls and the source code of the application will have to be available in order to collect provenance since provenance is collected during source code compilation.

3.1.7 Provenance from Log Files: a BigData Problem

Goshal et al [16] 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 [15] 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 instumentation 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 com-

ponent 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. [25] 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-aactuator I/O operations which may be used to ensure trust of connected devices.

3.1.10 Backtracking Intrusions

Samuel et al [24] 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 [18] 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 a 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 does not demonstrate causality and dependency between objects.

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 [7] developed a system which allows maximizing provenance storage collection in the Linux OS environment using mandatory access control (MAC) policy. This enables 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 [36] 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 [5] 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 Provenance Data Compression

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

Hussein et al. [21] 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 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 is sensor id represented as characters in which arithmetic coding compression technique is applied to. Our approach look at not just the who of provenance but also all other aspects of provenance is considered.

Chapter 4

Sensor Data Provenance Collection Framework for the IoT

In this chapter, we introduce a novel approach for how provenance is collected and modeled along the IoT architectural stack. We also propose a design and implementation of such an IoT provenance collection framework. This section defines a model for relaying provenance in IoT systems which is built on top of PROV-DM.

4.1 IoT Provenance-Collection Framework

A provenance model is used to represent causal dependency between objects and it is usually represented as a Directed Acyclic Graph(DAG). This representation facilitates visualization of provenance relationships and offers a unified format for representing provenance data. We chose PROV-DM as the model to represent provenance for our implementation because it allows for the proper representation of all of the relationships in which we envision for IoT devices. From the IoT architecture as illustrated in Figure 4.3, data is disseminated from sensors and actuators across layers of the IoT architectural stack. The provenance data produced from various sensors and actuators are collectively aggregated at the gateway or the cloud layers. We translate provenance data to PROV-DM format at any layer of the IoT stack. This allows processing of provenance at all layers even in the case of a network failure. Figure 4.1 illustrates the provenance data aggregation across the layers of the stack. Data is collected from sensors and actuators aggregated, and passed along the hierarchy.

Using the scenario of a smart home as illustrated in chapter 2, a detailed example of

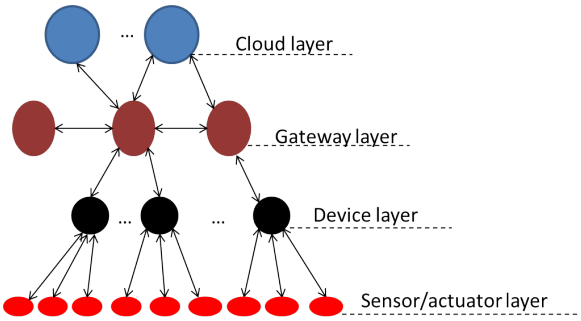

Figure 4.1: IoT provenance-collection data aggregation

how our provenance collection framework can be applied as follows:

- Provenance data is collected from sensor and actuator readings of devices contained in the smart home (e.g thermostat, refrigerator, and smart doors). The provenance data is collected as specified in a policy document.

- Provenance data from multiple sensor and actuator readings collected from each device are aggregated and passed along to the gateway, which transmits it to the cloud for long-term storage and use in data analytics. Each layer in the provenance IoT architecture is independent of other layers and maintains provenance information that can be mapped using the PROV-DM format which allows for the representation of dependencies between sensor-actuator readings contained in the device. This allows for analysis at each layer independently.

4.1.1 Provenance-Collection Model Definition

Since PROV-DM provides a generic model for representing provenance information, we define a more specific model for representing provenance data in IoT devices based on PROV-DM. In the context of IoT provenance, we define entity, process and agent as follows:

- Agent: An agent is any data object that is responsible for the actions of an activity. An agent could also be an entity or an activity. Examples of agents in an IoT

architecture are sensors, actuators, user roles(e.g admin). A unique identifier is given to each agents contained in an IoT framework.

- Entity: An entity can be defined as a data object that contains information which can be modifiable. An example entities are device files, processes, device memory contents and network packets. An Entity is identified by an id and can include additional attributes such as location and time.

- Activity: An activity is an action that an agent makes on an entity. Examples of activities are basic file access operations such as read, write, delete, update, memory access such as load and store, and network activity such as send and receive.

To better explain the provenance-collection model, a use case is illustrated in Figure 4.2, which depicts a model for a smart home.

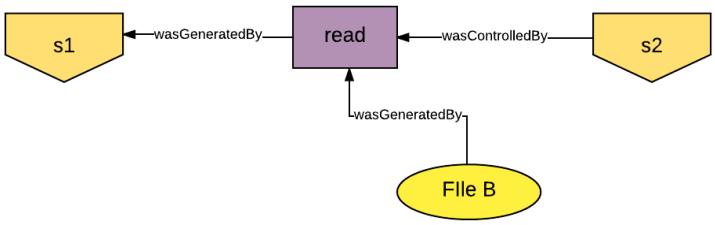

Figure 4.2: Provenance-model use case

Figure 4.2 depicts a dependency relationship between two sensors, s1 and s2. s1 is a smart thermostat which regulates the temperature of the home and s2 is a sensor that detects the temperature of the environment. s1 checks the temperature of the environment in order to regulate the temperature of the house accordingly. s1 tries to access sensor temperature information from s2. According to the provenance data model definition, s1 and s2 are agents. The activity performed on s2 by s1 is read. Sensor data from s2 is stored in File B which is an entity that s1 tries to read sensor readings from s2. The relationship between components in the model is illustrated on the edges between types

in the provenance model. We use the same relations contained in PROV-DM to represent relationships between types contained in the IoT framework.

4.2 Provenance-Collection System

In this section, we show how components of our system and describe how provenance trace is collected across the IoT framework. Figure 4.3 displays the system architecture of our approach. Sensor and actuator readings in the form of input and output (I/O) events are recorded by the tracer component. This component intercepts system level I/O events and produces trace information represented in Common Trace Format (CTF). CTF encodes binary trace output information containing multiple streams of binary events such as I/O activity. Trace information is converted to provenance data in the PROV-DM IoT model and serialized to PROV-JSON. CTF conversion to PROV-DM will be achieved using babeltrace. This conversion can happen at any layer of the IoT stack. 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 for efficient storage, query and visualization of provenance data.

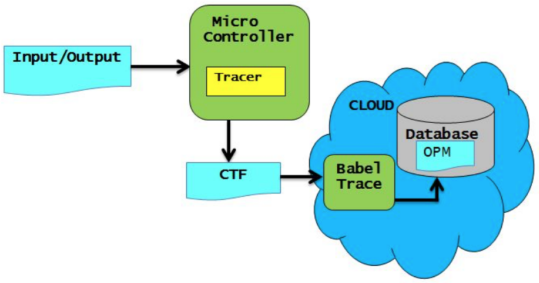

Figure 4.3: Proposed model

Our goal is to create a provenance-aware system which records I/O operations on data

for devices connected in an IoT system. For our implementation, several tools and hardware components are utilized in the development of our prototype, outlined below:

- BeagleBone Black Board. This device is a microcontroller used to evaluate our approach. We choose the BeagleBone Board because it is a representation of what can be found on an IoT gateway device and it has the capability to include custom hardware in programmable logic. Also, BeagleBone is a low cost, simple IoT demonstrator that was chosen for its high performance, onboard emulation, and IoT gateway projects can be programmed without additional need for hardware tools.

- Neo4j, a graph database which allows optimized querying of graph data. Since provenance represents causal dependencies, it is ideal to use a graph database to store the relationships between objects

- lttng, a software tool for collecting system level trace on Linux system.

- Babeltrace: This is a trace converter tool. It contains plugins used to convert traces from one format into another.

- barectf: This is an application that collects bare metal application trace in CTF.

- yaml generator: yaml generator creates yaml files. A yaml configuration file contains information on what barectf application needs in order to generate CTF trace output. This consist of configuration settings such as an application trace stream, packet type, payload type and size.

4.3 Experiment Evaluation

I plan to evaluate the effectiveness of my approach for provenance collection by using an intrusion detection system specifically developed for IoT. An IDS is used to detect malicious attacks using either a rule-based or an anomaly-based approach. A rule-based approach

allows for intrusion monitoring by looking for specific known signatures of malicious attacks. An example of a signature-based IDS is an anti-virus software. An anomaly-based IDS identifies an intrusion by checking for patterns that falls out of the normal system behavior. Most anomaly-based approaches make use of machine learning to classify normal or anomalous behavior. These approaches can be useful to detect previously unknown malicious attacks.

We propose a provenance-based IDS for IoT that extends Provenance-aware Intrusion Detection and Analysis System(PIDAS) [37], which uses system-level provenance data to provide real-time vulnerability intrusion detection on system behaviors. This system collects provenance from system calls. PIDAS contains three essential components: collector,detector and analyzer. The collector records provenance information of applications running in the user space. Provenance data is stored in a key-value pair database for easy query of acyclic graphs. The detector extracts dependency relationships from the provenance data and stores these relationships in a repository (BerkleyDB) for further analysis. The analyzer is responsible for identifies possible intrusion activities by making queries to view all dependencies from the suspected intrusion point. Provenance detector is made possible by using a provenance-based algorithm that matches the path contained in the graph.

The detector consists of three steps. Rule-built collects normal system behavior of provenance data and stores this information in a ruleDB. A ruleDB consists of a key-value pair which has parent and child relationships in the dependency graph. The rule-built is used to match observed provenance events to detect system intrusion. Provenance information in the form of a dependency relationship is matched with the child-parent dependency rule information contained in the ruleDB. This information is used to compute the decision value which determines if there has been an intrusion. A description of the PIDAS rule matching and scoring algorithm is described in detail below:

- provenance data is divided into dependency relationships, $Dep_1, ..., Dep_n.Dep_i = (A, B)$. Where A is the parent of B

- The scoring algorithm checks if there exists a match in the path between edges contained in the provenance dependency database, ruleDB. If there exists a path, a score of 1 is given to the path. This score is known as the path dubiety. Otherwise, if the path does not match any edge in the ruleDB, a score of 0 is given to the path.

  Let R = provenance information of a program and G = provenance information contained in ruleDB. For each $Dep_i = (A, B) \in R, if Dep_i = (A, B) \in G$. Path dubiety = 1 else set intrusion dubiety = 0

- The path decision, P is the sum of the dubiety of all the edges contained in the provenance graph divided by the number of edges in the provenance dependency database.

$$P = \frac{\sum_{i=1}^{W} dubeityof Dep_i}{W}$$

  P is compared with a threshold value, T. If P > T, an anomaly exists in the system.

I plan to compare our proposed framework using PIDAS with a baseline which consists of an implementation of PIDAS on an embedded system. More specifically, I plan to evaluate the false negative and true positive rate for the IDS system as compared with the baseline. False positive dictates the number of times in which an intrusion has been wrongly detected. This signifies that a path contained in an applications provenance was detected and not found in the ruleDB. True positive rate identifies the number of times a intrusion has been detected. I also plan to evaluate the throughput of the system by evaluating the overhead incurred using the system.

Chapter 5

Storage Optimization Framework for Provenance Data Storage in IoT

In this chapter, we optimize storage of provenance using our IoT provenance-collection framework.

5.1 A Policy-Based Approach for Provenance Storage

Provenance easily can generate more data than the size of the entity's data about which provenance is being collected. Due to the large influx of real-time data generated from sensors and actuators in IoT, we anticipate large amounts of provenance data will be generated from our provenance collection framework. The resource constraints on the sensor-actuator and the device layer of the IoT architecture makes storing that data even more challenging. To address this challenge, we define a policy driven framework that provides a policy scheme for the optimization of provenance storage and addresses resource constraints encountered when storing large amounts of provenance data. Policy allows flexibility of provenance collection and pruning for a specific IoT application thereby eliminating irrelevant provenance data. Since organizations have varied provenance requirement(s), making a static provenance policy is not suitable for the IoT. Using policy for storage optimization is slightly different from the typical use of policy in enforcing access control.

For the IoT framework, the product manufacturer is considered a policy creator. This reduces the complexity of creating and managing policy documents by the IoT device consumers. A policy creator is a user which has the right to add, delete or modify a policy

document.

A policy document is required to specify which provenance data to collect thereby providing storage access to provenance unlike system access control which evaluates user access rights to view a resource.

The policy framework consists of a policy engine. The policy engine contains authorization and enforcement components that provides and enforce decisions on how provenance data should be stored.

A policy document is a component of the policy framework. It identifies provenance data that is considered relevant to the IoT application.

Our policy architecture is modeled using the Common Open Policy Service (COPS) Standard [20]. COPS consists of components for policy generation, evaluation and enforcement. The Policy Enforcement Point (PEP) enforces decisions received from the Policy Decision Point (PDP). The PDP evaluates policies and generates decision based on the evaluation. The model can be extended to include a secondary decision point (SDP) which allows for distributed policy evaluation, thus freeing up the PDP from communication bottlenecks caused by large amounts of requests received by a single PDP.

Figure 5.1 below illustrates the system architecture of our proposed policy-based storage framework. Different layers of the IoT architecture contain different decision and enforcement components. The sensor-actuator layer of the IoT architecture is omitted because it has negligible memory resources and the sensors and actuators are usually physically part of a device in the device layer and as such does not have any data to prune.

Policy document which is generated by the policy creator serves as an input to the PDP component and is evaluated at the device, gateway and cloud layer. The PEP which is involved with generating requests is located in the device and gateway layer. SDPs can be located in the gateway layer, which allows for policy evaluation without inuring additional network overhead of communicating with the PDP located in the cloud layer. To implement our policy based framework, We plan to use the eXtensible Markup Language (XACML) [1] and generate an enforceable provenance storage policy for our IoT provenance-collection

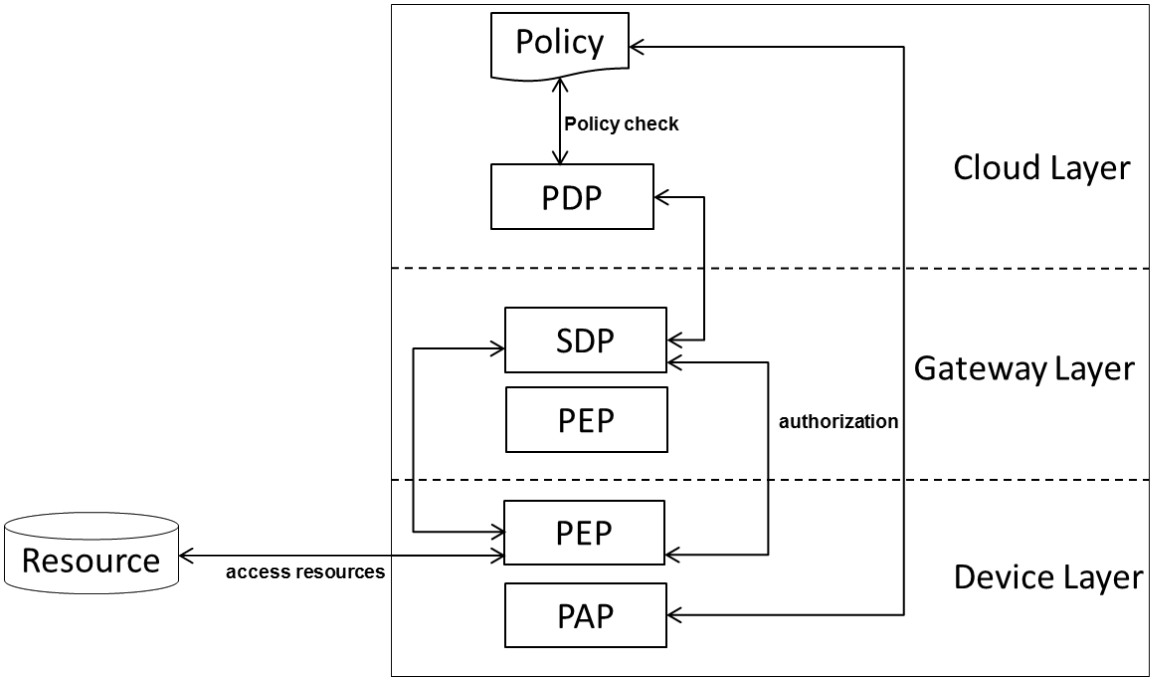

Figure 5.1: Policy based system architecture which allows for effective data storage of provenance data

framework. We extend XACML to include the SDP component.

Using the use case of the smart home depicted in chapter 2, a policy framework could be implemented and incorporated into the IoT which allows a device administrators to specify what kinds of provenance data to collect. The policy acts an enforcement point providing an efficient storage mechanism in a resource constrained environment. The contents of the implementation details for the policy framework which specifies the policy grammar and the policy architecture across the IoT stack are left as future work.

5.2 Provenance Information Flow

This section unifies components of our IoT provenance collection system. Figure 5.2 depicts a flowchart which illustrates how provenance is generated and how the storage policy is

used to enforce access on provenance data. Input data are represented in green while processes are represented by the blue rectangles. Policy document determines what kinds of provenance data to collect and store. The policy creator can be a device owner who has access and authority to the device. A yaml configuration file is passed as an input to the the tracer component. This contains the barectf application that collects CTF trace from sensor-actuator readings on the device. The yaml configuration is an essential portion of the tracer system. It is used to generate application CTF trace output. It contains instructions of what trace data to collect. The policy framework takes a modular approach: policy component can be added at various layers of the IoT architecture. The policy engine is involved with the authorization and enforcement of policies generated. This portion generates a decision based on the policy evaluation. The response from the policy allows provenance effectively pruned. Provenance data stored as CTF trace in neo4j, a graph database and is mapped to PROV-JSON format. The PROV-JSON serialization is used as input to our IDS system which serves as the provenance application. Figure 5.2 below illustrates the relationship between various component in the provenance aware IoT framework.

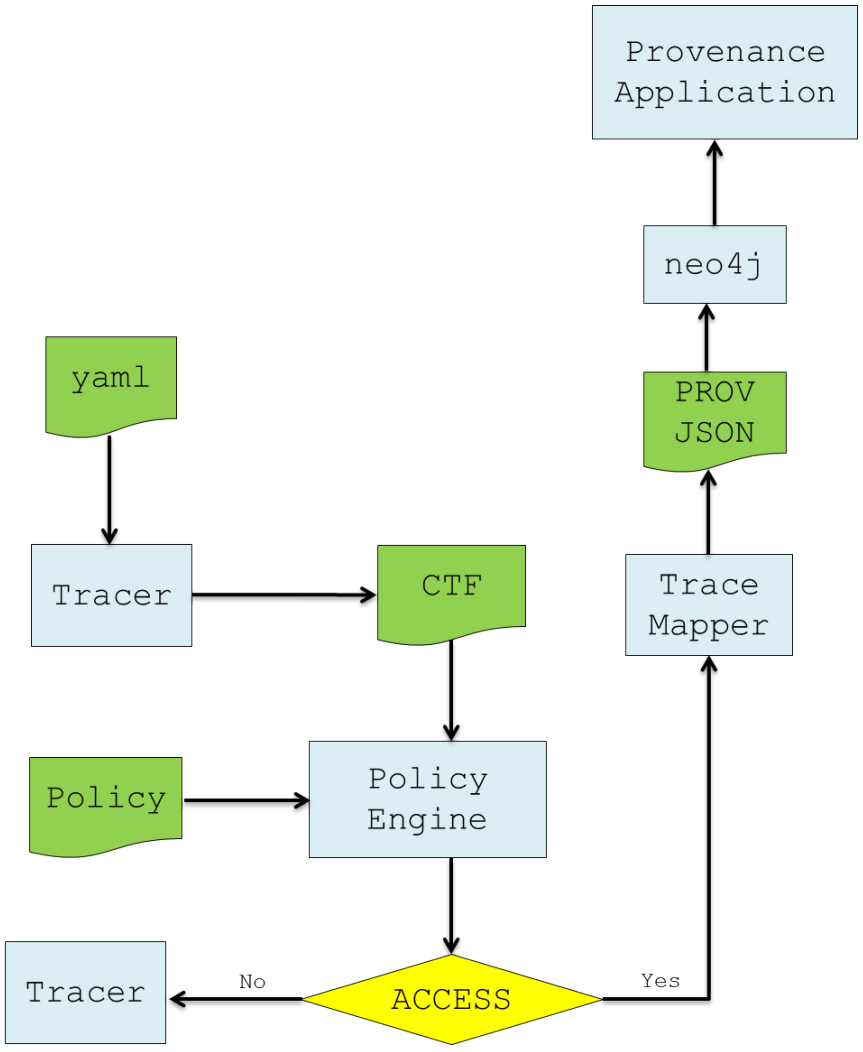

Figure 5.2: Provenance Information Flowchart

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 sensore-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 [19] are needed to ensure the confidentiality, and integrity of provenance data.

References

[1] eXtensible Access Control Markup Language (XACML) Version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

[2] PROV-DM: The PROV Data Model. `https://www.w3.org/TR/prov-dm/`.

[3] A General-Purpose Provenance Library. 2012.

[4] Prov-dm: The prov data model. Technical report, W3C, apr 2013.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Adam Bates, Dave Tian, Kevin R. B. Butler, and Thomas Moyer. Trustworthy whole-system provenance for the linux kernel. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 319–334, Berkeley, CA, USA, 2015. USENIX Association.

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

[11] 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.

[12] 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.

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

[14] Dave Evans. The internet of things how the next evolution of the internet is changing everything. Technical report, CISCO, Apr 2011. Accessed: 2016-10-01.

[15] 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.

[16] 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.

[17] 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.

[18] Paul Groth, Simon Miles, and Luc Moreau. PReServ: Provenance recording for services. In *in Proc. AHM05*.

[19] 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.

[20] Shai Herzog. The COPS (Common Open Policy Service) Protocol. RFC 2748, March 2013.

[21] 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.

[22] 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.

[23] David Reinsel John Gantz. The digital universe in 2020: Big data, bigger digital shadow s, and biggest grow th in the far east. Technical report, EMC Corporation, apr 2012.

[24] 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.

[25] 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.

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

[27] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The Open Provenance Model Core Specification (V1.1). *Future Gener. Comput. Syst.*, 27(6):743–756, June 2011.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

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

[34] Mark Stanislav. Hacking iot: A case study on baby monitor exposures and vulnerabilities. Technical report, Rapid7, sep 2015.

[35] 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.

[36] 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.

[37] Yulai Xie, Dan Feng, Zhipeng Tan, and Junzhe Zhou. Unifying intrusion detection and forensic analysis via provenance awareness. *Future Gener. Comput. Syst.*, 61(C):26–36, August 2016.

[38] Shams Zawoad and Ragib Hasan. Fecloud: A trustworthy forensics-enabled cloud architecture. 2015.