HOWARD UNIVERSITY
GRADUATE SCHOOL
Department of Electrical Engineering and Computer Science

DISSERTATION COMMITTEE

APPROVED:

Legand Burge III, PhD.

Charles Kim, Ph.D.

Danda Rawat, Ph.D.

Ebrima Ceesay, PhD.

Gedare Bloom, PhD.
Dissertation Advisor

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

HOWARD UNIVERSITY

Trace-Based Data Provenance For Cyber-Physical Systems

A Dissertation

Submitted 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

by

EBELECHUKWU NWAFOR

Washington, DC
May 2018

Acknowledgements

Like the saying goes, “it takes a village to raise a child”. This journey would not have been possible without the help of so many people. First and foremost, I would like to thank the one true ominipotent being, God for making this possible, without him i will not be where i am today. I would also like to thank my parents, Mr.Benneth and Mrs. Chinwe Nwafor for their constant encouragement, love and support. I would like to thank my advisor Dr. Gedare Bloom for his guidance and encouragement throughout this process, for believing in my research ideas and also for his constructive critique and valuable input in various drafts revisions of my dissertation. I would like to thank my committee members Dr.Rawat Danada, Dr.Legand Burge III, and Dr.Charles Kim for their insightful comments and feedback. I would like to thank my God mother Dr.(Mrs) Ifeoma Obiwuru for her unwavering love and support throughout my academic career and for also encouraging me to reach greater heights. I would like to thank some of the most amazing mentors i have been fortunate to have: Dr.Adedoyin Adeyoiga, and Dr. Deivy Peterescu. You all are a source of inspiration and a beacon of light. You both have inspired me to be relentless while I undertake this journey of completing my doctorate degree. I would also like to thank Howard University and the National Science Foundation (Grant No. CNS-1646317) for being so generous in funding my research. Finally, I would like to thank everyone whose name i might have not mentioned that have played a part in the successful completion of my doctoral dissertation.

Abstract

Cyber physical systems (CPS) have revolutionized the way humans interact with computing devices by coupling process automation with networked services leading to increased productivity and ease of life. However, privacy and security problems arise as a result of CPS connectivity, heterogeneity, and complexity. One major issue is data trust—how do we ensure that data generated from these devices have not been compromised by malicious actors? Data provenance offers a solution to this question of data trustworthiness by maintaining and tracking dependency and causality among data objects, which can then be used as a tool in detecting malicious attacks.

In this dissertation, we explore the application of data provenance to device security in the CPS ecosystem by way of a provenance-collection framework for CPS that uses lightweight software tracing via application instrumentation. Problematic to automatic and thorough provenance collection is the overhead of storing complete historical metadata for all data objects. To address this problem, we investigate the use of data pruning algorithms to discard non-essential provenance metadata from streaming traces prior to their aggregation and conversion to provenance. We evaluate the effectiveness of provenance collection using a provenance anomaly-based intrusion detection system (IDS) in a climate control system and an automotive domain. Additionally, we evaluate the effectiveness of pruning algorithms in the automotive CPS application domain.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cyber-physical systems (CPS) and the Internet of Things (IoT) are rapidly accelerating the pace of evolution in the production of embedded systems. Spanning the range of personal devices for recreation and comfort, critical-care medical implants and monitors, intelligent transportation systems, and large-scale industrial facilities, embedded systems underlie the fabric of modern society and are commonplace in nearly all aspects of human daily activities. The benefits of widespread adoption include advances in financial management, health and welfare, economic output and productivity, and recreation. By and large, these benefits derive from distributed physical environment sensing and actuation integrated with data analytics. Ubiquitous deployment of embedded systems with access to Internet gateways has led to the exponential growth of the IoT, which consists of heterogeneous, multiscale CPSs. Unfortunately, the rapid connection of these systems to external networks increases their attack surface and remotely exploitable vulnerabilities introduce new attack vectors that can have disastrous financial and physical impacts. In 2010 Stuxnet [16]—a sophisticated self-replicating virus—was discovered that successfully exploited vulnerabilities in industrial control system (ICS) software. In 2015, security researchers demonstrated a remote exploit targeting a Jeep vehicle that allowed the attacker to completely control the compromised automobile over the Internet [21]. In 2016, an exploitable vulnerability was discovered that enabled Internet-connected smart thermostats to be attacked by remote ransomware [44]. Such exploits can have devastating and long-lasting impact on individuals, institutions, and even nation-states.

A major security challenge for CPSs is ensuring trust in data. Data are trustworthy in case malicious agents have not tampered with the source or transformation of the data.

A means of ensuring such trust is through data provenance, which identifies both the origin of data and their lineage through the history of data transformation. The record of provenance reveals intricate dependencies among data objects that enables auditing their production and use. Thus, data provenance addresses a key component of Lampson's "gold standard" of security [31]. As a tool for security, data provenance is used in forensic analysis [51, 33] and intrusion detection systems (IDS) [15, 54]. The inherent difficulty in provenance collection is managing the growth of provenance, which increases each time a new data object is created or an existing one is manipulated; most important, even if one data object replaces another data object, the provenance records contain the source and lineage information about both. In systems with limited storage, such as embedded systems in a CPS, this difficulty is keen.

This dissertation introduces *Prov-CPS* as a framework for provenance collection in the kinds of resource-constrained embedded system devices found in fielded CPSs. Prov-CPS uses a streaming approach to achieve time-efficient generation of provenance, and prunes streaming provenance at the originating device for economic memory use. The construction of provenance is divided in two distinct phases by Prov-CPS: first, application source code is instrumented to generate traces of events relevant to an application's data use, and second the traces are converted to a provenance graph representation suitable for use in provenance analysis tools. This two-phase, trace-based approach is better suited to CPS software than prior provenance collection frameworks [42, 38, 9], which collect provenance through operating system event monitoring (i.e., system calls and file accesses) that is impractical for resource-constrained embedded systems that often use baremetal software or a minimal real-time operating system without the necessary facilities or available computation and memory to implement such frameworks. Prov-CPS does not require an operating system. An ontological provenance data model [1] provides a unified semantic for representing provenance across heterogeneous devices and applications, which enables flexible mechanisms for application-defined control over both time and space resource utilization. We demonstrate and evaluate Prov-CPS using an anomaly-based IDS that analyzes prove-

nance graphs.

Research Contribution

This dissertation makes the following contributions:

1. **Prov-CPS: trace-based provenance collection framework.** The design and implementation of a provenance collection framework for capturing trace-based events for tracking data flow in an application's logic in memory constrained bare-metal devices (i.e., devices without an operating system). This framework generates a trace of streaming events through application instrumentation with low execution run-time overhead that is decoupled from the more resource-intensive activities of provenance processing and analysis.

2. **Provenance data model for trace events.** A unified data model that represents the information flow semantics of data movement among components in an abstract CPS. This abstraction is mapped to real CPSs to facilitate automation of provenance data collection and analysis.

3. **Storage optimization through trace event pruning.** Support within the Prov-CPS framework for data pruning algorithms that discard events from streaming trace data prior to their conversion to provenance. These algorithms can be tailored to the needs of the provenance analyst yet take advantage of application domain knowledge.

4. **Experimental evaluation of Prov-CPS.** Evaluation of the efficiency of Prov-CPS in space, with and without pruning, using a provenance-based IDS that identifies anomalies in data provenance records.

Organization of Dissertation

The remaining portion of this dissertation is organized as follows: In chapter 2, we review background information on CPS, anomaly detection, and data provenance. We also discuss PROV-DM, a provenance ontological construct. In Chapter 3, we outline related work on provenance collection systems, provenance-based pruning techniques, and provenance-based intrusion detection. In Chapter 4, we give a detailed description of Prov-CPS, and provenance-trace data model. In Chapter 5, we describe our pruning heuristics. In Chapter 6, we discuss our anomaly detection algorithm using provenance graphs. Finally, we conclude in Chapter 7 with future research directions.

Chapter 2

Background

In this chapter, we describe the concept of data provenance.

2.1 Data Provenance

Provenance is defined as "The place of origin or earliest known history of something" [43]. However, provenance and lineage are used interchangeably to denote the origin of something and transformation(s) that occurred to it over time. The notion of provenance originates from the art world, where an art piece auctioned is accompanied by a paper trail that denotes the artwork's chain of custody. This trail allows a buyer to ascertain the authenticity of the artwork. Likewise in computing, provenance provides a holistic history of data transformations that occur on a data object from inception to its present state.

Data provenance can ensure data integrity [12] and establish causality between data objects through information-flow tracking. Characteristic information provided by provenance include the who, where, when, and what of data transformation. **Who** identifies an actor responsible for a particular transformation of a data object. An example of "who" in a CPS use case is a sensor, which is identified as the ultimate progenitor of values derived from its sensor readings. **Where** provides the location of a transformation, for example a geolocation for the sensor. **When** associates time with a transformation, e.g., a timestamp obtained when a particular sensor reading was taken. **What** denotes the kind of transformation, such as a *read* operation or a *send* operation.

Provenance has been applied to database management to track the derivation of a query, in forensic analysis to determine what activities led to a system event [11, 35, 55],

in scientific experimentation for experiment reproducibility [18, 14, 6, 40], and in intrusion detection [51, 15, 24] to detect what activity might be responsible for a malicious compromise.

The increase in applications of provenance has led to provenance collection systems to support them. For example, within the context of the operating system, PASS [38] and HiFi [42] are two Linux-based provenance collection systems developed to capture interactions between files and system call events. In the scientific community, Chimera [18] and myGrid [40] collect provenance to support scientific experiment workflow reproducibility. Notably, most of these applications are tailored for large-scale, memory-intensive applications that are difficult to adapt directly to memory-constrained CPS devices.

2.2 Model for representing data provenance

To facilitate the creation of cross-platform provenance, an abstract representation of provenance typically uses a directed acyclic graph (DAG) in which vertices correspond to data and edges to interactions between data. A selection of vertex types and edge relationships constitutes a provenance model. The W3C standardizes a provenance ontology and a provenance data model, PROV-DM, to enable interchangeable provenance across heterogeneous systems. We adopt notions from the PROV-DM in the creation of Prov-CPS.

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 2.1 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.

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

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

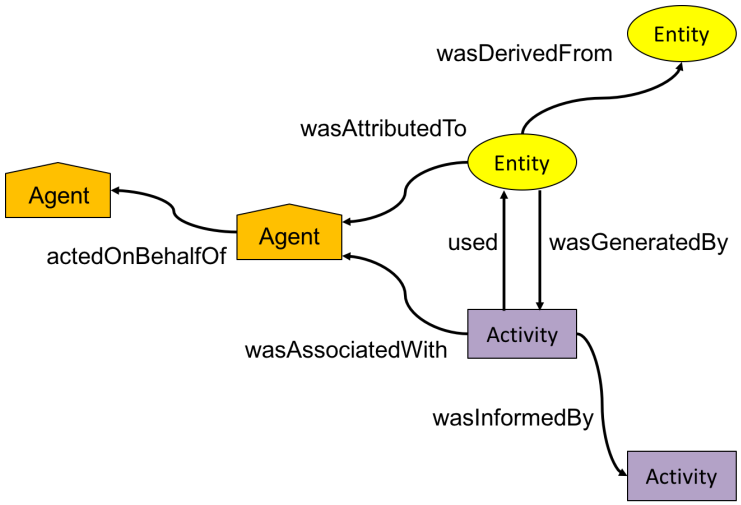


Figure 2.1: Prov-DM respresentation showing types in the model (Entity, Activity, and Agent) and the relationships between them

2. used: An entity generated by one activity has been adopted by another activity.

3. wasInformedBy: Signifies the exchange of an entity by two activities.

4. wasDerivedFrom: Represents a copy of information from an entity.

5. wasAttributedTo: Denotes relational dependency between an entity and an agent when the activity that created the agent is unknown.

6. wasAssociatedWith: An agent created or modified the activity.

7. actedOnBehalfOf: Delegation of authority from an agent to itself or another agent to perform a particular responsibility.

Using the terminology of PROV-DM, we formally define a provenance graph as a labeled directed acyclic graph, $p = (V, E)$ where V is a set of vertices $V = \{v_1, ..., v_n\}$ such that $v_i = (type, value)$, with type one of the core types of PROV-DM, and E is a set of edges $E = \{e_1, ..., e_n\}$ where $e_i = (v_s, v_d, label)$, with v_s, v_d as source and destination vertices, and $label$ is one of the PROV-DM relations. Two vertices v_x, v_y are equal (denoted $v_x = v_y$)

if $v_x.type = v_y.type$ and $v_x.value = v_y.value$. Two edges e_x and e_y are equal (denoted $e_x = e_y$) if $e_x.v_s = e_y.v_s$, $e_x.v_d = e_y.v_d$, and $e_x.label = e_y.label$. We use the union operator $\cup$ over edge sets in the usual way of the union of sets.

Chapter 3

Literature Review

This section discusses the state of the art in data provenance collection systems, pruning techniques for data provenance, and provenance-based intrusion detection systems.

3.1 Data Provenance Collection Systems

There has been a considerable amount of work done on data provenance collection [10, 19, 36, 39]. 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 CPS. Some of the prior work done on data provenance collection are outlined as follows:

3.1.0.1 Kernel-space provenance collection

Muniswamy-Reddy et al. [38] developed PASS, 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 addresses application level provenance for memory constrained embedded systems without requiring OS modification.

Pohly et al. [42] developed Hi-Fi, a 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.

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. This approach differs from our approach since it deals with provenance collection of system call events in the kernel space.

King and Chen [30] 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. This approach deals with provenance collection of OS based system events. Our approach deals with provenance collection on application based event derived through application instrumentation on CPS devices.

3.1.0.2 User-space provenance collection

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. Our approach focuses on event flow of application logic on memory constrained CPS devices without requiring OS level interference.

Spillance et al. [45] 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 Berkeley 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. In our approach to provenance collection, we are particularly interested in provenance collection of application event logic on memory constrained CPS devices.

Ghoshal and Plale [20] 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. This approach differs from our approach since our approach focuses on provenance data from application event logic in real-time as opposed to post processing of log files.

3.1.1 Provenance collection 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 application based events which are used to ensure trust of connected devices.

3.1.2 Provenance collection for scientific experiments

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. Our approach is focused on collecting provenance data on application events in CPS devices through application instrumentation.

3.1.3 Application Instrumentation-based provenance collection

Tariq et al. [46] 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:

1. The application is compiled and converted into bitcode using the LLVM C compiler, clang.

2. The LLVM Tracer module is compiled as a library.

3. LLVM is run in combination with the Tracer, passed to include provenance data.

4. Instrumentation bitcode is converted into assembly code via the compiler llc

5. The instrumented application and assembly code is linked into an executable binary and sent to the SPADE kernel.

In this approach, provenance collection is done at program function level during function entry and exit which might lead to collecting a lot of information that might be irrelevant. Our approach on the other hand specifies what events to collect via application instrumentation.

Gessiou et al. [19] 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. Our approach differs from this approach since this approach allows for instrumentation of kernel level trace. Our approach focuses on application level event trace.

ProvThings [49], an IoT based provenance-collection framework and API, allows application developers to generate provenance through instrumentation of devices by identifying security sources and sinks through a selective instrumentation algorithm. This framework's implementation is focused on a Samsung-based IoT platform, SmartThings. ProvThings

utilizes a policy-based approach to intrusion detection in which a provenance graph pattern is matched based on defined system behavior. One distinct difference between this approach and our approach is that ProvThings is focused on API based cloud devices connected to a hub. Our approach deals with provenance collection on memory constrained CPS devices.

ContextIoT [28] is an access control framework that provides contextual integrity support for IoT applications by identifying sensitive actions and run-time prompts. Contextual information is defined based on data flow of the execution code during run-time and utilizes a rule-based approach to application security by defining permissible actions to sensitive resources based on data flow. A prototype was implemented on Samsung's SmartThings IoT platform. Our approach differs from the previously mentioned IoT-based provenance collection systems since it provides provenance collection at the device level as opposed to smart device APIs and cloud-based components.

3.2 Provenance-based Pruning

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. [47] 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. This approach to pruning relies on the MAC policy scheme in the Linux OS. Our approach deals with pruning with consideration to memory constrained CPS devices which might not contain an OS.

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 Wireless Sensor Network, 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 this approach 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.

Another common approach looks at ways to compress provenance graphs. Xie et al. [53] adapt a modified web compression technique for provenance graphs based on locality, similarity, and consecutiveness of nodes contained in the provenance graph. Wang et al. [48] proposed a dictionary based compression technique for provenance graph compression. Xie et al. [50] proposed a hybrid approach which consists of web and a dictionary compression technique. This approach focuses on compression as a technique to storage optimization. Our approach employs the use of data pruning heuristic to eliminate provenance data that is unnecessary to our application use case.

3.2.1 Provenance-based IDS

PIDAS [52] is an IDS that uses provenance graphs generated from system calls, which reveals interactions between files and processes. This approach employs the use of a rule-matching technique for anomaly detection. Our approach on the other hand involves the vectorization of provenance graphs and a similarity technique to compare graphs. We also use a threshold value to classify anomalous instances.

FlowFence [17] is a rule-based approach to malicious intrusion detection that allows users to declare the data flow of sensitive data in an application thereby data that does not fit the predefined flow path are denied access to resources. This approach uses a rule-based technique to anomaly detection. Our approach uses a threshold score which can be fine-tuned to determine anomalous instances.

PANDDE [15] uses OS-based provenance to track a data object downloaded from a database after which it profiles a user's behavior based on predefined actions such as data printing, emailing, and storage to detect data exfiltration attacks. This approach differs from our approach to anomaly detection since it utilizes a rule-matching technique in which a user profile is directly matched to currently executing program in the detection phase.

FRAPpuccino [24] is a provenance-based fault detection system that runs on a cluster of machines which detects anomalous behavior in the provenance graph based on a dy-

namic sliding window and clustering. One drawback of this approach is assumed notion that anomalous instances can be detected from a subset of provenance graphs thereby using sliding window which only considers a subset of the entire training and test set. FRAPpuccino shares some similarity with our approach since they utilize a window size for anomaly detection however some key differences exists. We utilize an approach to anomaly detection which reduces a provenance graph into a vector space representation and determine anomalous system events based on a defined threshold value.

Mukherjee et al. [37] proposed a precedence graph based anomaly detection technique to detect message injection attacks in J1939 networks. Precedence graphs are used to model temporal relationships between data events. To detect anomalous instances in the precedence graphs, normalized flux capacity is used. Flux capacity is the product of in-degree and out-degree of a node. We utilize a different approach to anomaly detection. Our approach reduces provenance graphs to a vector space representation which is then compared (training and test graph set) using cosine similarity metric.

Hailesellasie and Hasan [23] introduce a graph-based IDS for industrial controllers that relies on exact comparison of graphs derived from control logic of a programmable logic controller program. Our approach does not use exact comparison, because exact graph comparison is brittle in complex, irregular CPS applications.

3.3 Summary of Related Work

While exploring prior state of the art on provenance collection systems, it is evident that none of the previous work addresses the issue of provenance collection on cyber-physical systems with major emphasis on memory constrained bare-metal devices (i.e., devices without operating systems). This dissertation addresses the issue of provenance collection on memory constrained cyber-physical systems.

For provenance-collection systems, [28, 49, 46] are closely related to PROV-CPS. We compare PROV-CPS to the closely related work as follows:

1. ProvThings [49] and ContextIoT [28] generates an abstract syntax tree of an instrumented program at runtime after which a selective code instrumentation is performed based on selecting security sensitive methods. This process incurs additional overhead. PROV-CPS generates application trace events which are converted into provenance. To further optimize task, both processes (i.e., generating trace events and trace to provenance conversion) can be performed in parallel.

2. Tariq et al. [46] utilizes the LLVM compiler which generates provenance for all function calls entry and exit points contained an application before it is then optimized. This produces a lot of provenance data irrelevant to the application. Our approach specifies what trace data to collect from program instrumentation at run-time before our pruning heuristic is used to further streamline trace data generated. PROV-CPS deals beyond the function level. It deals with interactions within events in an application from a variable to a more complex data structure.

A closely related work to the use of anomaly detection applied in the J1939 network bus is the technique proposed by Murkherjee et al. [37]. In the experimental evaluation, one key difference is the use of a fixed window size of 1 seconds to generate a precedence graph from the dataset. Our approach on the other hand utilizes varied window sizes. This allows for better understanding of the effects of false positive and true positive rate on the window size. Another difference is that a window is classified as malicious only if it contains at least a quarter of malicious messages.This might be problematic if there exists less than a quarter malicious messages in the window. In our approach, we classify a window as malicious if it contains a malicious message. In addition, malicious messages are injected at three phases: the start, the middle and towards the end of the dataset. In our approach, we inject messages at time gaps that are 12 times the average time gap which allows a realistic and thorough simulation of malicious messages without time overlap.

Chapter 4

Provenance-Aware CPS (Prov-CPS)

In this chapter, we explore the integration of provenance within the CPS ecosystem. We introduce Provenance-Aware CPS (Prov-CPS), a provenance collection framework for CPS devices. We evaluate the effectiveness of our framework by developing a prototype system for proof of concept.

4.1 Prov-CPS Design

The first step to realizing the benefits that provenance offers for CPS is to develop a framework that collects provenance data from embedded systems in the CPS automatically and with minimal system overhead. Such a framework should be

1. **Complete:** A system must collect all provenance of causal dependencies between events. That is, provenance data collected must be complete in such a way that any datum can be backtracked to its source.

2. **Abstract:** A system must be generic enough such that it can be reused across multiple CPS applications and domains.

Our approach leverages application-defined traces of data creation and transformation to achieve completeness, and uses a predefined provenance data model that is modular and generic to analyze those traces and convert them to provenance in a abstract manner.

4.2 Trace-based Provenance Data Model

Embedded systems in a CPS application comprises of sensors and actuators operating on a continuous basis with data periodically created, transformed, and transmitted through device network interfaces. Properly representing these operations requires a data model that encompasses the intricate data flow within, between, and across devices. This model should be interoperable for varied application domains.

Although provenance models are well-studied, no existing provenance model properly represents the dataflow of application logic for embedded systems in a CPS. To address this issue in Prov-CPS, we define a provenance-trace model that maps from the W3C's PROV-DM ontological representation to concepts relevant for a trace-based approach to provenance collection in a CPS. This mapping enables interoperability of Prov-CPS among heterogeneous systems and applications. Relationships between each component of provenance-trace model is depicted using relations as described in PROV-DM [1]. Table 4.1 describes the mapping of terminology between the Prov-CPS trace-based provenance model and PROV-DM.

Table 4.1: Concept mapping from trace-based provenance to PROV-DM.

Concept Description	**Provenance-trace model**	**PROV-DM**
Data to track with provenance	Data Object (d)	Entity
Time at which provenance is produced	Timestamp (t)	Entity
Identity accountable for operations on data	Controller (c)	Agent
Identity of component that creates, transforms, or transmits data	Producer (p)	Agent
Operations that create, transform, or transmit data	Action (a)	Activity

The Prov-CPS trace-based provenance model consists of five core components: data object, timestamp, controller, producer, and action. We define a data object as the information created, used, manipulated, or transmitted by a CPS. An action is work performed on a data object, such as creation, modification, or transmission, and a producer conducts

the action on the data object. A timestamp represents when an action occurred. A controller manages the actions of a producer; in the CPS setting at the embedded system level, the controller is the computing device. A data object might be a result of another data object's modification by a producer, or an aggregated value including multiple data objects, where the aggregated value is contained in a transformed data object with a provenance relationship to the data objects included in its aggregation. The software that calculates the aggregate is the producer.

Consider the periodic temperature readings taken by a thermostat as a simple example mapping between an application domain and the trace-based provenance model. In this example, the thermostat itself is the controller, the data object is a temperature reading, the producer is a temperature sensor, the action is reading of the sensor, and the timestamp is the thermostat's notion of time when the reading was taken.

The key to effective collection of provenance is the definition of a general trace event format that is applicable to multiple applications and varied operations on data. We define a trace event record as a tuple

$$event_{trace} = (t, d, a, c, p)$$

where t is a timestamp, d is a data object, a is an action, c is a controller's identifier, and p is a producer's identifier.

For every trace event, Prov-CPS establishes the following fixed relationships between the provenance model components:

1. c *used* p
2. (t, d) *wasGeneratedBy* a
3. p *wasAssociatedWith* a
4. $(t, d)_p$ *wasAssociatedWith* (t, d)

where $(t, d)_p$ is the most recent (t, d) generated by an action associated with the producer p. This last relationship establishes a temporal relationship between data objects generated by the same producer.

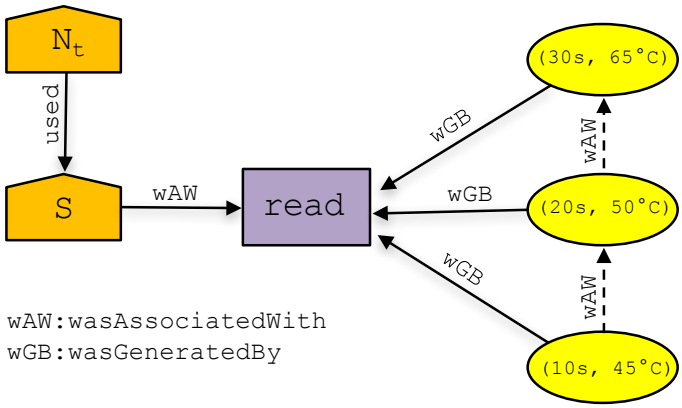


Figure 4.1: Provenance graph generated from a thermostat device.

4.2.0.1 Example: Thermostat

We demonstrate the Prov-CPS data model with an example of thermostat device (N_t) containing a sensor (S) that generates temperature readings at a time interval of 10 seconds. Temperature readings of the first 3 consecutive data objects are 45 °C, 50 °C, and 65 °C respectively. Trace events generated for these first three data objects are

$$event_{trace1} = (10s, 45°C, read, N_t, S)$$

$$event_{trace2} = (20s, 50°C, read, N_t, S)$$

$$event_{trace3} = (30s, 65°C, read, N_t, S)$$

Relations between components of the trace event are formed as described earlier. N_t forms a *used* edge with S, which itself forms a *wasAssociatedWith* edge with the *read* action. Each temperature reading's timestamp and data object are grouped as a pair (t, d) and form a *wasGeneratedBy* edge with the *read* action and a *wasAssociatedWith* edge with

the previous pair associated with S through the *read* action that generated it. Figure 4.1 depicts the provenance graph for this thermostat example.

4.3 Prov-CPS Architecture

Prov-CPS uses a modular approach that decouples the act of tracing events from that of linking them to form provenance. In this section we describe the architectural design of this approach, and defer implementation details to Section 4.4. Figure 4.2 illustrates the overall system design consisting of four primary components:

Tracer is a program located on the CPS device that is responsible for generating trace data from application-level data events.

Trace mapper converts trace data into a provenance graph representation using the ProvCPS data model in which trace events are mapped to provenance and relationships between them are established.

Graph database is used to store provenance data for further processing. A graph database such as Neo4j is an ideal choice because it allows for easy graph traversal and querying analytics.

Provenance application uses provenance data for specific tasks such as anomaly detection or digital forensics.

4.4 System Implementation

Figure 4.3 depicts our implementation of the architectural components of Prov-CPS. For the tracer component we use **barectf** [3], which is a program for tracing applications written in the C programming language specifically meant for execution on bare-metal devices (devices without an operating system). We chose this program is lightweight, portable, and allows for implementation on memory constrained embedded systems. Barectf generates trace functions that can be called from an application's source code. The event field

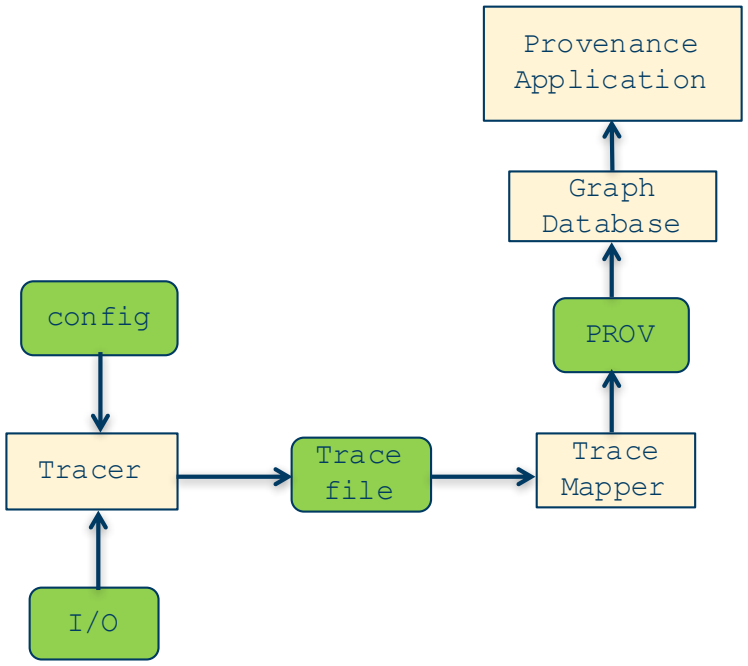


Figure 4.2: Prov-CPS Conceptual System Design.

traced corresponds to values of variables or functions in the application source code that are specified as parameters of the trace function. Event field specification is done by a using a configuration file written in the **yaml** language from which the trace functions are automatically generated by barectf. At execution time, the trace functions generate trace output to a stream object in **common trace format** (CTF) [4] packets. CTF is a fast binary trace format for representing trace data.

We implemented a trace mapper on top of the babeltrace [2] python bindings for programmatically manipulating CTF streams. With babeltrace, we are able to access trace packets contained in a CTF packet stream and convert them to a provenance graph representation. To generate provenance graphs, we utilize the **PROV** python library for creating W3C compliant provenance ontology documents. The provenance graph generated is serialized into an intermediate file in JSON format that can be stored in Neo4j for efficient query and graph processing.

The provenance application we use to evaluate Prov-CPS in this work is a graph-based

anomaly detection algorithm for use as a CPS intrusion detection system, which we describe in chapter 6.

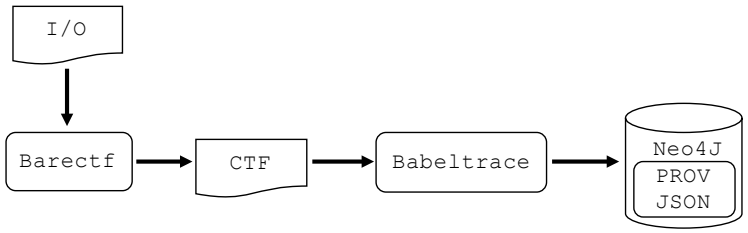


Figure 4.3: Prov-CPS System Implementation.

4.4.1 Complexity analysis

Tracer appends trace data to a list of events. This takes $O(1)$ time and $O(n)$ space. **Trace mapper** iterates through all of the events and maps them to PROV-O using Provenance-trace model. This takes $O(n)$ time and $O(n)$ space.

4.5 Summary

This chapter introduces PROV-CPS, a provenance collection framework for CPS devices. Components of PROV-CPS such as tracer and trace mapper are discussed with system implementation details. The next chapter discusses storage optimization techniques for provenance data generated in PROV-CPS.

Chapter 5

Storage Optimization for Trace Data

In this chapter, we explore optimization of provenance graph size using pruning approach.

5.1 Introduction

A major challenge faced by any provenance capture system is that provenance data incurs additional storage overhead [13]. That is, provenance can easily generate more data itself than data about which provenance is being captured. This storage problem could lead to rapid decline in system performance both in collection and analysis of provenance. Prov-CPS is no exception. Furthermore, the resource constraints imposed on CPS devices further exacerbates the challenge. Table 5.1 outlines trace and provenance data generated for increasing counts of trace events of a sample application that generates temperature data.

Table 5.1: Trace data size as number of trace increases

Trace Events	**Trace Size (KiB)**
200	8.2
500	25.6
1000	51.2
1500	76.8

5.1.1 Trace event cost estimation

To verify the hypothesis that there exists a linear growth in trace and provenance data generated, we provide formulas to estimate the size of provenance and trace events data generated by a device. To calculate the total storage size of trace data generated on a device, we estimate the size of an event. Every event consists of a finite set of event fields such a controller, producer, action.

The total size of a trace data can be calculated using the function:

$$T_{size} = N_{events} \times K$$

where T_{size}, N_{events}, represents trace size, and number of events respectively. K represents the sum of the size of all of the fields contained in an event trace:

$$K = t_{size} + d_{size} + a_{size} + c_{size} + p_{size}$$

Additionally, given the size of a device, D_{size}, the number of trace events that can be stored on a device, $D_{capacity}$, is derived using the function:

$$D_{capacity} = \frac{D_{size}}{K}$$

Figure 5.1 illustrates the graph of the estimated and actual trace generated from the initial analysis in Table 5.1. For calculating the cost of trace events data, the size of each component contained in the trace event is set to 10 bytes. We can observe a linear growth between the estimated and generated data for trace events.

Motivated by the need to provide an optimized storage for provenance data generated, we address the issue of device storage at the trace level by pruning events based on predefined heuristics. Pruning is defined as eliminating unwanted events in a trace to optimize storage space.

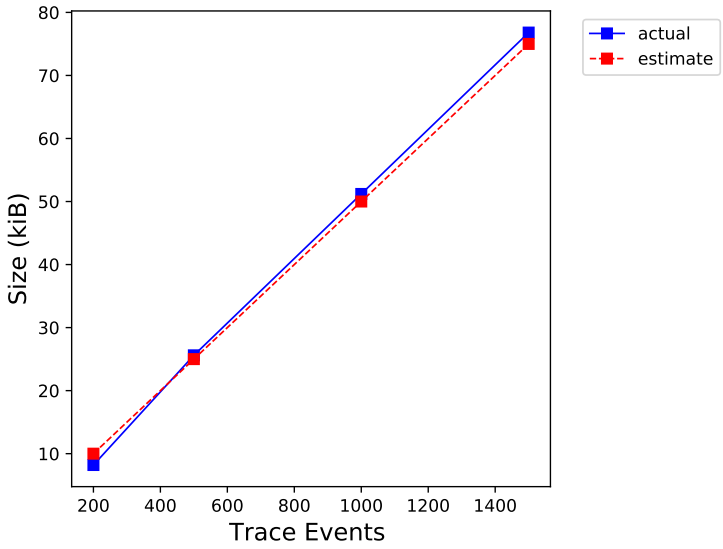


Figure 5.1: Storage growth per trace events.

5.2 Types of Provenance Pruning

To determine what data to prune is a challenging task and often application dependent. Based on the current state of the art, the common approaches to data pruning are:

Pruning at collection removes provenance during or before the collector stores it. For example, an application might decide to eliminate repeated trace data as they are generated, or a system-call provenance collector in an operating system kernel might ignore user configuration and temporary files that are unnecessary for the provenance application use case.

Pruning after collection involves removing unwanted provenance information after provenance has been saved to a storage system.

Policy-based pruning [7] involves the use of user-specified conditions (expressed as a policy) to determine data to prune. Policy offers a flexible method of pruning, but requires a robust policy language to express policy decisions, which often could be a challenge to design. A policy is evaluated by a decision point that returns accept or deny.

Compression [53, 26, 48] involves reducing the size required to represent a data value. Provenance compression may be applied to the provenance or to its graph structure. The use of lossy or lossless techniques can be employed in provenance pruning, but the existing literature advocates for lossless approaches.

Graph summarization involves aggregating graph nodes and edge relationships based on similar attributes without loosing information contained in the original graph. This reduces the size of the original graph while preserving the structural pattern of node and edge relationships. The structural pattern of provenance graph consists of homogeneous nodes (i.e., entity, agent, activity) which can be harnessed to provide attribute-based summarization to an original graph structure. Summarization improves graph visualization and allows for better understanding of graph structures and relationship between various node and edge interactions.

5.3 Pruning in Prov-CPS

Our approach involves pruning at collection by selectively dropping events from the set of captured traces. We introduce two techniques for provenance pruning: first-in, first-out (FIFO) and prioritized.

1. **FIFO pruning:** FIFO is the naïve approach that uses the natural order of arrival time to select events to discard. As trace events are generated they are added to a FIFO. Once the queue has reached its capacity, each new trace event causes the oldest trace event in the FIFO to be eliminated. An advantage of FIFO pruning is that it can be implemented without application knowledge. A disadvantage is that it is not selective in what it discards, and if provenance becomes more important (for a given provenance application) as it becomes older, then FIFO is not a good choice.

2. **Prioritized pruning:** The prioritized approach assigns a priority value to each event as it is traced based on an application defined priority, thus it extends the definition

of the trace event with an additional field for the priority value. Although application knowledge is required to determine the priority of each event, many CPS application domains include natural priority values that can be immediately leveraged by the provenance application. With this approach, lower priority events are discarded from the set of trace events first. A drawback of this approach is the cost needed to sort trace events when there are many priority values.

5.3.1 Complexity analysis

FIFO Pruning is implemented using a queue data structure. This takes constant time $O(1)$ to insert and delete events from queue. Since there are n elements contained in the queue, the worst case space utilized is $O(n)$. **Priority-based pruning** is implemented using a heap data structure. It takes $O(\log n)$ to insert and delete elements contained in the heap with a space complexity of $O(n)$.

5.4 Summary

This chapter motivates the need for storage optimization in PROV-CPS. Two pruning heuristics, FIFO and Priority-based pruning, were proposed to address the issue of the storage overhead generated by PROV-CPS. The next chapter introduces a provenance-based anomaly detection system for the detection of anomalous instances through information flow of device events.

Chapter 6

Provenance-Based Anomaly Detection

This chapter discusses the application of provenance data for intrusion detection of malicious events in a CPS device. 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 graphs derived from an application's normal execution. The result is an anomaly score which is compared with a previously set threshold to classify an observed provenance graph as either anomalous or benign. We evaluate the effectiveness of our approach with a sample IoT application that simulates a climate control system.

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.

6.1 Graph Similarity

Similarity 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, a widely used similarity

metric in information retrieval that has been shown to perform well with sparse datasets. 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 between vectors, we are only concerned with the positive values bounded in [0,1]. The cosine similarity between two vectors, X and Y, is computed by:

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

In order to apply cosine similarity between provenance graphs, we compute a vector representation which reduces the graph into an n-dimensional vector space where n represents the total number of edges contained in the union of all edge sets. Figure 6.1 illustrates the vector space conversion of provenance graphs. $\boldsymbol{G_1}$, and $\boldsymbol{G_2}$ which consists of vertices $A, B, E, F, G, I, J, L, S, R$ and edge labels $used, wAW, wGB, wDB$. The vector space representation of u_1 is the occurrence of edges contained in the edge set of graph G_1, which are also found in the collective union of edge sets. Algorithm 1 further outlines the concept of graph to vector conversion.

6.2 Anomaly Detection on Provenance Graphs

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 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.

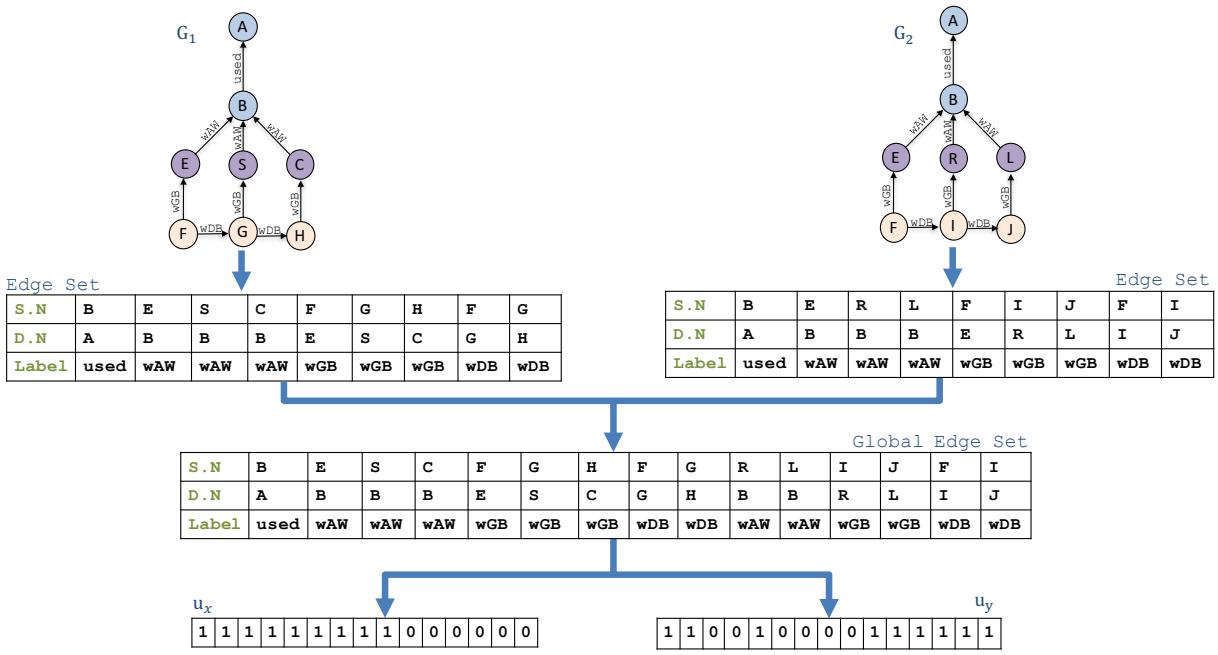


Figure 6.1: Provenance graph conversion to vector space. u_x, u_y represents vectors generated from both provenance graphs and labels are an abstraction of $(type, value)$ tuple.

Many CPS 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. For example, in a thermostat application, temperature readings generated might be converted from Celsius to Fahrenheit and utilized as feedback to an actuator. Each iteration of a control loop sequence generates a path in a provenance graph. This notion can be leveraged to define an expected provenance graph for each application.

The expected regularity of provenance graphs in CPS applications motivates a supervised learning approach to anomaly detection. This approach consists of two phases: observation phase, also known as the training phase, and the detection or test phase. In the observation phase, the system collects provenance data considered to be a representation of the normal system behavior. In the detection phase, the provenance graph set is compared with the provenance graph derived from subsequent observations to determine if an anomaly exists by measuring similarity between this graph and the provenance graph set.

Algorithm 1 Graph to vector conversion.

```
1: procedure GraphtoVector(E, E_G)
2:     n ← |E_G|
3:     Q[k], Q[i] ← 0, 0 ≤ i < n
4:     for e_j ∈ E do
5:         for e_g ∈ E_G | 0 ≤ g < n do
6:             if e_j = e_g then
7:                 Q[g] ← Q[g] + 1
8:             end if
9:         end for
10:    end for
11:    return Q
12: end procedure
```

Note that provenance graphs from the observation and detection phase form a graph set. A global edge set, E_G represents the union of edge sets contained in a graph set. Algorithm 2 is the graph anomaly detection function given an observation phase graph set, P, and a detection phase graph, G. Z represents a list of the cosine scores from comparing each of the provenance graphs in the observation phase graph set with a detection phase graph. The maximum cosine similarity score of elements contained in Z is taken as the score for the detection phase graph, and if that score is above the threshold T then the graph is considered normal, otherwise it is classified an anomaly.

6.2.1 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 often is manually set to a value that is defined by domain experts.

Algorithm 2 Detection algorithm given an observation phase graph set, P, a detection phase graph, G, and a threshold T.

procedure GRAPHANOMALY(P, G, T)
  INPUT: $P = \{G_0, ..., G_n\} \mid G_i \leftarrow (V_i, E_i), 0 \leq i \leq n.$
  $E_G \leftarrow \cup_{i=0}^{n} E_i$
  $G \leftarrow (V, E)$
  $Q \leftarrow GraphtoVector(E, E_G)$
  $Z \leftarrow \{\}$
  for $G_i \in P$ **do**
    $N_i \leftarrow GraphtoVector(E_i, E_G)$
    $z \leftarrow Cosine\_Similarity(Q, N_i)$
    $Z \leftarrow Z \cup z_i$
  end for
  $s_{val} \leftarrow max(Z)$
  if $s_{val} \geq T$ **then**
    return normal
  end if
  return anomaly
end procedure

For automatic anomaly threshold detection, an approach would be to determine the average of a set of cosine similarity scores. Additionally, one can use prediction methods such as regression, decision trees, or random forest to define an anomaly score. Threshold values can be increased or decreased to alter the anomaly detection accuracy.

6.2.2 Complexity analysis

In Algorithm 1, each edge contained in edgelist E is iterated over all of the edges contained in the global edgelist E_G. The algorithm takes $O(|E| \times |E_G|)$ and $O(|E| +$

$|E_G|$) in time and space complexity respectively. In Algorithm 2, graph G is compared with every graph contained in the set P. This involves a call to `GRAPHTOVECTOR` and `Cosine_Similarity` functions and produces a time complexity of $O(|P| \times |E_G|^2)$ and $O(|P| \times |E_G|)$ respectively. Therefore, the overall time complexity of Algorithm 2 is $O(|P| \times |E_G|^2)$ and runs with a space complexity of $O(|P| \times (|E_G|))$.

6.2.3 Threat Assumptions

Due to the ubiquitous nature of CPS 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. It is important to note that we do not provide security using cryptographic primitives such as encryption or digital signatures. Additionally, this framework does not prevent attacks on the modification of data within the normal data flow.

6.3 Summary

This chapter introduces an anomaly detection algorithm for detecting anomalous system behavior as depicted by the data flow of events in a CPS device. Detailed information on the steps taken such as converting a graph into a vector representation and determining the similarity between vectors are discussed. The next chapter evaluates the effectiveness of PROV-CPS using the anomaly detection algorithm discussed in this chapter which is applied to a climate control system and an automotive domain.

Chapter 7

Experiments

In this chapter, we evaluate Prov-CPS using our anomaly detection algorithm on a climate control system and an automotive network bus.

7.1 Climate Control System

The experiment evaluation serves as a preliminary study to confirm the correctness of our theoretical approach in detecting anomalous instances between provenance graphs. We evaluate our intrusion detection algorithm on Prov-CPS by implementing an application which simulates a climate control system. Climate control systems ensure a proper functional environment for people and machinery. Constant irregularities in temperature could have devastating effects on industrial machinery. This system consists primarily of a heating ventilation and air conditioning (HVAC) system which uses temperature and humidity data to regulate environmental conditions within a building. We utilize a publicly available dataset [32] 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 generated at a period of fifteen minutes. We utilize the temperature and humidity data as input to our simulation. We generate provenance graphs for each week of the year. We compare the provenance graph generated in consecutive weeks to see how they differ using our graph similarity approach (i.e., week 1 compared to week 2, week 2 compared to week 3 etc.). In order to generate a tuple representation of events (t, d, a, c, p) using the trace-based provenance data model as described in section 4.2, we used information such as temperature and humidity data, activity performed, i.e., read , sensor and device identifier information.

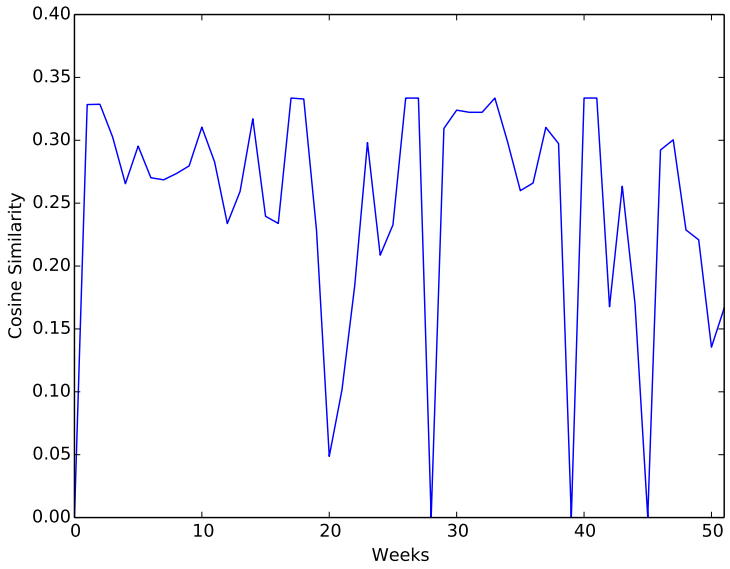


Figure 7.1: Provenance graph comparison of climate control system by week.

Figure 7.1 depicts the cosine similarity between provenance graphs generated from the first occupant by preceding weeks.

Since the dataset does not contain attacks, the declines shown in Figure 4.3 would likely cause false positives.

7.2 J1939 Network Bus

We evaluate Prov-CPS using J1939 CAN data recorded from a Peterbilt truck [5]. This dataset consists of J1939 messages representing normal driving conditions observed on the high speed CAN bus that were logged for a duration of 74 seconds consisting of a total of 16,223 messages. The J1939 messaging protocol is an extension of the CAN messaging protocol which is mostly used in heavy-duty trucking vehicles and commercial buses to communicate between in-vehicle electronic control unit (ECUs). A J1939 message consists of a message ID, number of data bytes sent, data, and error checking code. A J1939 message ID consists of a 29-bit value used to differentiate messages sent by different ECUs. For our

purposes, we only rely on two fields contained in the message ID: the source address field, which denotes the origin of the message, and the priority field to support priority-based pruning.

From the J1939 dataset, we recreate a trace where each message causes one trace event. We map the source address to the producer identifier in the Prov-CPS model, and map the CAN bus as the controller. The action is a "receive" (Rx) activity denoting a passive listener on the bus. We also have the ability to recreate the timestamps from the dataset to be the same in the trace events, but in these experiments we do not use the trace event timestamp. The data object of each trace event consists of the (up to 8) data bytes of the J1939 message. The entire 29-bit message ID is stored as a priority field in the trace event to use for priority-based pruning.

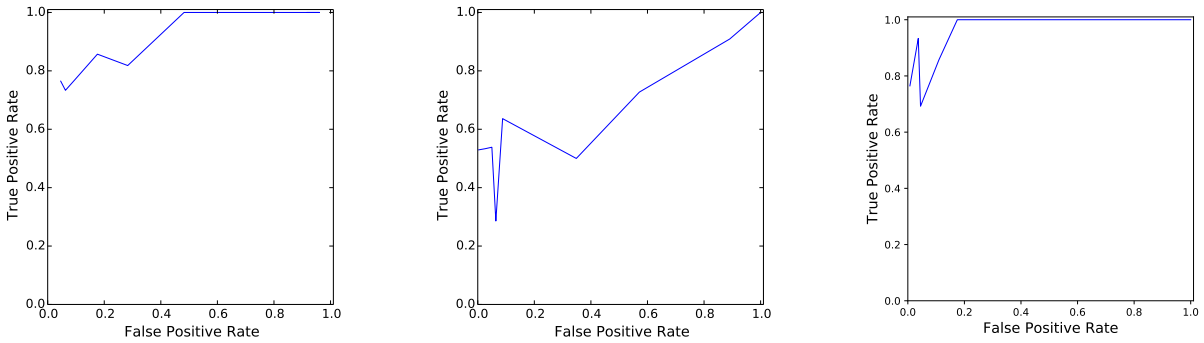
Based on the enormous physical constraint that exists in performing real-time message injection attacks on heavy vehicles, and the support of prior work that uses simulation of message injection attacks in CAN networks [29, 41], we chose to simulate message injection attacks on a CAN bus network. This attack is achieved by recreating the expected log file containing CAN messages received on a CAN bus in an event of a malicious attack. Our message injection attack was modeled after the attacked performed by Mukherjee et al. [37]. We simulate message injection attacks by injecting malicious messages into the J1939 dataset using a TSC1 message code, which has a message ID beginning with 21 bits all 0. These messages are used to control the engine's RPM or torque. We inject 10 consecutive TSC1 messages at varied points in the dataset within an existing time gap between two consecutive messages that is greater than 12 times the minimum time gap between two consecutive messages. This is to ensure that we accurately simulate malicious message injection without either time overlap or in excess of the bus bit rate. We chose 10 different log files with the injected attacks in distinctly different regions of the log file (at least 400 messages apart).

For anomaly detection as described in Section 6, we take the first quarter of the driving log as a training (observation) set. We do not inject any new messages in this portion of

the log, and therefore it is the same in all experiments. The remaining three quarters of the driving log are the test (detection) set, and each anomalous log file contains injected attack messages somewhere in that set. We further divide the training and test sets into equal-sized windows, where the window size is a fraction f of the number of messages in the training set, that is, each window has size equal to $f * n/4$ where n is the number of messages in the log file, i.e., 16,223. A provenance graph is generated for each window, and so the window size is varied as a parameter to explore the effect that scaling graph size has on the performance of the anomaly detector. Performance is measured by collecting the number of true negatives (TN), true positives (TP), false negatives (FN), and false positives (FP) and calculating the true positive rate (TPR) and false positive rate (FPR) in the usual way as:

$$TPR = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{FP + TN}$$

For each window size, i.e., for each different value of f, we provide a cumulative TPR and FPR based on the sum of all the positives and negatives recorded over multiple variations of the log file.



(a) ROC with No Pruning (b) ROC with FIFO Pruning (c) ROC with Priority Pruning

Figure 7.2: Receiver operating characteristic (ROC) curves without pruning and with FIFO or Priority pruning.

Table 7.1 shows the cumulative results of anomaly detection summed over 10 log files

Table 7.1: Cumulative results of anomaly detection true negative (TN), true positive (TP), false negative (FN), and false positive (FP) for no pruning, FIFO-based pruning, and prioritized pruning based on application priority values.

Window Size	**No Pruning**				**FIFO Pruning**				**Priority Pruning**			
(Messages)	**TN**	**TP**	**FN**	**FP**	**TN**	**TP**	**FN**	**FP**	**TN**	**TP**	**FN**	**FP**
15	7310	13	4	353	7638	9	8	25	7607	13	4	56
20	5609	11	4	376	5829	8	7	156	5764	14	1	221
30	3605	10	3	382	3788	7	6	199	3808	9	4	179
40	2461	12	2	525	2795	4	10	191	2658	12	2	328
50	1713	9	2	676	2179	7	4	210	1971	11	0	418
126	492	10	0	458	619	5	5	331	591	10	0	359
253	138	11	0	331	201	8	3	268	108	11	0	361
506	9	11	0	220	25	10	1	204	0	11	0	229
1013	9	11	0	100	0	11	0	109	0	11	0	109

each containing 1 set of 10 injected messages. Each approach to pruning—no pruning, FIFO pruning, and priority pruning—was evaluated using the same input log files. For the pruning algorithms, we chose to prune one-half of the window size for these experiments. Each row shows a different window size, which corresponds to a different value of f ranging between 0.0039 to 0.25 with corresponding window sizes between 15 and 1013 messages. Cumulative results also are presented using a receiver operating characteristic (ROC) curve showing how performance changes with f. The ROC curve is a widely used metric for evaluating IDS systems. It is generated by plotting the TPR over the FPR. Figure 7.2 shows the ROC curve for each pruning algorithm (no pruning, FIFO pruning, and priority-based pruning) separately evaluated in the same way using the same modified datasets. As f increases, the window sizes get larger and so do the graphs, which increases their relative variability and therefore decreases the similarity scores, thus resulting in higher TPR and

FPR. With smaller f and therefore small graphs, the anomaly detection algorithm is better able to match the features of graphs in the training and testing sets, so TPR can stay high while FPR decreases; note that with smaller window sizes, there are also more decisions (positives and negatives) to make.

7.3 Summary

This chapter evaluates the effectiveness of PROV-CPS using an anomaly detection algorithm as described in chapter 6. The anomaly detection algorithm is evaluated on a climate control system and a J1939 network. In the climate control system, we compared the temperature values generated by preceding weeks. Preliminary result indicates the detection of anomalous irregularities in temperature data. On the J1939 network, we evaluated false positive and true positive rates based on a defined window size (f value) using a dataset which consists of malicious J1939 messages. The result indicates that the smaller the window size, the better the accuracy of detecting anomalous instances that exists in the dataset.

Chapter 8

Conclusion and Future Directions

8.1 Summary

This dissertation focuses on the integration of data provenance within CPS ecosystem for provenance-based anomaly detection. To achieve this, we developed a provenance-aware framework that collects the information flow of data dependencies within a CPS device in a non-intrusive fashion. Our approach leverages the PROV-DM ontology for portability and uses a software-based tracing mechanism to collect provenance without requiring operating system intervention. We address the issue of storage overhead by proposing a data pruning heuristic at the trace level which eliminates irrelevant data not required by our provenance intrusion detection system.

8.2 Future Research Direction

Some of the proposed future work are listed as follows:

1. **Security and Privacy protection of Provenance Data:** Provenance collection raises privacy issues. How do we ensure that the vast amount of data collected is not invasive to the privacy of device users. Privacy preserving techniques can be used to anonymize provenance data. Proper encryption and authentication techniques [25] are needed to ensure the confidentiality, and integrity of provenance data.

2. **Real-time Anomaly Detection:** We provide an offline anomaly detection using provenance graphs. Future work will include the modification of our IDS algorithm

to include detection of anomalous instances in real-time.

3. **A hybrid approach to storage optimization of trace data:** We will explore how the use of a combination of techniques such as compression and graph summarization in addition to our approach will further reduce the size of trace data generated.

4. **End to End integration of Prov-CPS:** Current implementation of Prov-CPS does not include an automated flow between various components. A more seamless framework is needed where all of the components of PROV-CPS operates without user interference.

5. **Anomaly detection using a machine learning approach:** Machine learning algorithms have been shown to effectively detect anomalous instances however, most of these algorithms are computationally intensive and incur additional memory overhead. It will be beneficial to explore the use of advanced machine learning algorithms such as neural networks, and support vector machines for anomaly detection using provenance data generated from our framework. In addition, it is important to determine what the maximum memory capacity is for integrating the aforementioned algorithms on CPS devices.

6. **CPS-Cloud integration** This dissertation focuses on provenance collection on end devices. In the future, we plan to explore the integration of provenance in the cloud. Specifically, we will explore how data is disseminated across end devices to the cloud and how this interaction can be further streamlined for optimal service execution.

References

[1] PROV-DM: The PROV Data Model, April 2013. Accessed: 2016-10-01.

[2] Babeltrace. `http://diamon.org/babeltrace/`, 2018.

[3] barectf. `https://github.com/efficios/barectf`, 2018.

[4] Ctf. `http://diamon.org/ctf/`, 2018.

[5] Peterbilt dataset. `http://tucrrc.utulsa.edu/J1939`, 2018.

[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 (Jing) Tian, Kevin R.B. Butler, and Thomas Moyer. Trustworthy whole-system provenance for the linux kernel. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 319–334, Washington, D.C., 2015. USENIX Association.

[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] Adam J. Bates, Kevin Butler, Andreas Haeberlen, Micah Sherr, and Wenchao Zhou. Let sdn be your eyes: Secure forensics in data center networks. In *Proceedings of the NDSS Symposium*, 2014.

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

[13] Uri Braun, Simson Garfinkel, David A. Holland, Kiran-Kumar Muniswamy-Reddy, and Margo I. Seltzer. Issues in automatic provenance collection. In Luc Moreau and Ian Foster, editors, *Provenance and Annotation of Data*, pages 171–183, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[14] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: Challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1345–1350, New York, NY, USA, 2008. ACM.

[15] Daren Fadolalkarim, Asmaa Sallam, and Elisa Bertino. Pandde: Provenance-based anomaly detection of data exfiltration. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16, pages 267–276, New York, NY, USA, 2016. ACM.

[16] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32.stuxnet dossier. `https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf`, 2011.

[17] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. Flowfence: Practical data protection for emerging iot application

frameworks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 531–548, Austin, TX, 2016. USENIX Association.

[18] 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.

[19] 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.

[20] 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.

[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] Muluken Hailesellasie and Syed Rafay Hasan. Intrusion Detection in PLC-Based Industrial Control Systems Using Formal Verification Approach in Conjunction with Graphs. *Journal of Hardware and Systems Security*, 2(1):1–14, March 2018.

[24] Xueyuan Han, Thomas Pasquier, Tanvi Ranjan, Mark Goldstein, and Margo Seltzer. Frappuccino: Fault-detection through runtime analysis of provenance. In *9th USENIX*

Workshop on Hot Topics in Cloud Computing (HotCloud 17), Santa Clara, CA, 2017. USENIX Association.

[25] 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.

[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] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z. Morley Mao, and Atul Prakash. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS'17)*, San Diego, CA, February 2017.

[29] M. J. Kang and J. W. Kang. A novel intrusion detection method using deep neural network for in-vehicle network security. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pages 1–5, May 2016.

[30] 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.

[31] B. W. Lampson. Computer security in the real world. *Computer*, 37(6):37–46, June 2004.

[32] 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.

[33] Jin Li, Xiaofeng Chen, Qiong Huang, and Duncan S. Wong. Digital provenance: Enabling secure data forensics in cloud computing. *Future Generation Computer Systems*, 37:259 – 266, 2014.

[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] Rongxing Lu, Xiaodong Lin, Xiaohui Liang, and Xuemin (Sherman) Shen. Secure provenance: The essential of bread and butter of data forensics in cloud computing. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 282–292, New York, NY, USA, 2010. ACM.

[36] 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.

[37] Subhojeet Mukherjee, Jacob Walker, Indrakshi Ray, and Jeremy Daily. A precedence graph-based approach to detect message injection attacks in j1939 based networks. 2017.

[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, 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.

[40] Thomas M. Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, R. Mark Greenwood, Tim J. Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. volume 20 17, pages 3045–54, 2004.

[41] Satoshi Otsuka, Tasuku Ishigooka, Yukihiko Oishi, and Kazuyoshi Sasazawa. Can security: Cost-effective intrusion detection for real-time control systems. In *SAE Technical Paper*, page 11, 2014.

[42] 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.

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

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

[45] 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.

[46] 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.

[47] 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.

[48] C. Wang, S. R. Hussain, and E. Bertino. Dictionary based secure provenance compression for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):405–418, Feb 2016.

[49] Qi Wang, Wajih Ul Hassan, Adam J. Bates, and Carl Gunter. Fear and logging in the internet of things. 2017.

[50] Yulai Xie, Dan Feng, Zhipeng Tan, Lei Chen, Kiran-Kumar Muniswamy-Reddy, Yan Li, and Darrell D.E. Long. A hybrid approach for efficient provenance storage. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 1752–1756, New York, NY, USA, 2012. ACM.

[51] 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.

[52] 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.

[53] Yulai Xie, Kiran-Kumar Muniswamy-Reddy, Darrell D. E. Long, Ahmed Amer, Dan Feng, and Zhipeng Tan. Compressing provenance graphs. In *3rd USENIX Workshop on the Theory and Practice of Provenance*, June 2011.

[54] 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.

[55] S. Zawoad and R. Hasan. I have the proof: Providing proofs of past data possession in cloud forensics. In *2012 International Conference on Cyber Security*, pages 75–82, Dec 2012.