ABSTRACT

KIM, DONGHOON. Securing Data Flows within Software Application Networks in Cloud Environments. (Under the direction of Dr. Mladen A. Vouk.)

Cloud computing has become an acceptable paradigm and more and more business migrates to it. As applications are integrated into cloud environments, they may compose into application networks that communicate with each other and share resources in order to achieve their goals. However, a continuing concern when such application networks form in part or fully in public clouds is security. This dissertation develops and assesses an approach for pro-actively securing networks of cloud-based workflows.

The model we propose assumes a secure monitoring control point that keeps a real-time projection of the application network data-flows and transactions. Operational network nodes (data transformation points) could be in a secure environment themselves, or they could be in a public cloud. Data flows among the network components, and to the outside, are monitored and assessed for trustworthiness, validated with respect to a known and tested operational profile, and data integrity is checked. An anomaly requires a decision as to whether the data-flows and transformations/computations should continue, and how that affects confidentiality, integrity and availability expectations and balance. If a run-time failure is declared, recovery may be initiated.

As part of current work, we used the Kepler workflow model and implementation as the baseline framework. Kepler allows a variety of data-flow processing models - from fully parallel process networks, to sequential workflow models. We have augmented the standard Kepler provenance collection capabilities with a security analysis package intended to assess security of application network input/output flows. In order to understand threats that may face an application network that may operate in a cloud, we have done an extensive assessment of vulnerabilities reported in typical cloud SaaS and PaaS layers, and we have done a comprehensive analysis of Kepler run-time libraries and components. We show that under reasonable assumptions three gating functions - data integrity checking, input validation and remote access vetting - can provide a satisfactory
assurances regarding in workflow security. Our prototype secure Kepler workflow platform allows implementation of a variety of security countermeasures, and can serve to assess - in the set-up and testing phases - security of an application network that can be represented in the Kepler model.
Securing Data Flows within Software Application Networks in Cloud Environments

by
Donghoon Kim

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2016

APPROVED BY:

________________________________________  _______________________________________
Dr. Rada Chirkova                         Dr. Nagiza Samatova

________________________________________  _______________________________________
Dr. Henry E. Schaffer                    Dr. Mladen A. Vouk
Chair of Advisory Committee
DEDICATION

To my wife - Haegun Faith Jeon,
daughters - Uiiin Judy Kim, Uijin Joann Kim, and
parents - Kyouyoung Kim and Jeonghee Lim

The Lord is my shepherd; I have all that I need - Psalm 23:1
Hope does not disappoint us - Romans 5:5
BIOGRAPHY

The author was born in Gangneung, South Korea. He earned his Bachelor’s Degree at Gangneung-Wonju National University, S. Korea. After he received a master degree at Auburn University, USA, he worked at Samsung Electronics, S. Korea as a software engineer. During his PhD degree, he has conducted a variety of research on security in cloud computing systems, software engineering, and parallel computing as well as industrial projects at SAS Institute and IBM as an intern.
ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my advisor, Dr. Mladen A. Vouk, for his guidance and tremendous support. Also, I would like to thank my advisory committee members, Dr. Rada Chirkova, Dr. Nagiza Samatova, and Dr. Henry E. Schaffer, for their insight and assistance.

I would like to thank my faculties and staff in the department of Computer Science. I also thank Dr. Andy Rindos for his support while working at IBM as an intern.

I also thank my church members, Mahanaim, and my prayer list members. I give praise and glory to God, the Father of our Lord Jesus Christ, with all my heart.
# TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................ vii

LIST OF FIGURES ...................................................................................................... viii

Chapter 1 Introduction ............................................................................................... 1
  1.1 Problem and Motivation ......................................................................................... 1
  1.2 Security .................................................................................................................. 2
  1.3 Cloud Computing .................................................................................................. 4
  1.4 Application Networks and Workflows ................................................................. 6
  1.5 Approach ............................................................................................................... 7
  1.6 The Structure of Dissertation .............................................................................. 7

Chapter 2 Model ........................................................................................................ 8
  2.1 Introduction ........................................................................................................... 8
  2.2 Elements of Model ............................................................................................... 12
  2.3 Securing workflows ............................................................................................. 13
  2.4 Operational Profile .............................................................................................. 15
  2.5 Verification function ............................................................................................. 16
  2.6 Security Properties .............................................................................................. 18
    2.6.1 Threats ............................................................................................................ 18
    2.6.2 Input Validation .............................................................................................. 19
    2.6.3 Remote Access Validation .......................................................................... 19
    2.6.4 Data Integrity ............................................................................................... 20
  2.7 Validation process ................................................................................................. 22
    2.7.1 Adjudicator .................................................................................................... 23
    2.7.2 Access Control .............................................................................................. 23

Chapter 3 SaaS Security ......................................................................................... 24
  3.1 Introduction ........................................................................................................... 24
  3.2 Application ............................................................................................................ 25
  3.3 Network ................................................................................................................ 29
  3.4 Access Control .................................................................................................... 31
    3.4.1 Authentication ............................................................................................... 31
    3.4.2 Authorization ............................................................................................... 33
    3.4.3 Identity and Access Management ............................................................... 33
  3.5 Some Trends ......................................................................................................... 34

Chapter 4 PaaS Security ......................................................................................... 36
  4.1 Introduction ........................................................................................................... 36
  4.2 Background .......................................................................................................... 38
  4.3 Software Platform and Isolation ......................................................................... 40
  4.4 Virtualization ....................................................................................................... 42
# LIST OF TABLES

Table 4.1  An overview of major PaaS security concerns ............................................. 39

Table 5.1  A list of Scientific Workflow Management Systems (SWFMS) in broader use .... 54
Table 5.2  All the tables in the provenance-2.4 database ............................................. 58
Table 5.3  A security verification table in S-Kepler provenance database ......................... 62
Table 5.4  Security checking .......................................................... 84
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Illustration of a workflow management system</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>A simple workflow graph</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>An illustration of a workflow component structure</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Illustration of a workflow management system. Control plane and workflow engine are assumed to be secure.</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Illustration of a fundamental validation process for a data flow</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Security vulnerability categories and countermeasures of the Application Component</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Security vulnerability categories and countermeasures of the Network Component</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Security vulnerability categories and countermeasures of the Access Control Component</td>
<td>31</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Cloud service model with PaaS features</td>
<td>37</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>PaaS resources and services often rely on virtualization and encourage multi-tenancy. Resulting real and potential security threats are often mitigated through isolation.</td>
<td>50</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Trust assumption in the Kepler threat model</td>
<td>59</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>The Kepler System Architecture with the Security Analysis Package (K-SAP) added.</td>
<td>60</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>The overview of S-Kepler state transition to verify the security of Kepler applications</td>
<td>61</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>The timeline for processing data integrity</td>
<td>65</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td><strong>Input validation:</strong> Executing an external application by a command from an actor</td>
<td>67</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td><strong>Remote access validation:</strong> Accessing a remote database using a web service application</td>
<td>69</td>
</tr>
<tr>
<td>Figure 5.7</td>
<td><strong>Data integrity:</strong> Using local data</td>
<td>71</td>
</tr>
<tr>
<td>Figure 5.8</td>
<td>The execution result of a secure workflow with K-SAP</td>
<td>73</td>
</tr>
<tr>
<td>Figure 5.9</td>
<td>The execution result of an insecure workflow with K-SAP</td>
<td>75</td>
</tr>
<tr>
<td>Figure 5.10</td>
<td>A Kepler GUI showing a workflow with $f_i$ labels</td>
<td>80</td>
</tr>
<tr>
<td>Figure 5.11</td>
<td>Workflow validation</td>
<td>81</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>Kepler workflow management systems with K-SAP</td>
<td>86</td>
</tr>
<tr>
<td>Figure 6.2</td>
<td>Overhead in Kepler with PR and K-SAP</td>
<td>90</td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>A workflow to measure run-time overhead of K-SAP</td>
<td>93</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>Measuring execution time during a workflow run</td>
<td>96</td>
</tr>
<tr>
<td>Figure 6.5</td>
<td>Run-time overhead: Input validation and Data Integrity</td>
<td>97</td>
</tr>
<tr>
<td>Figure 6.6</td>
<td>Run-time overhead: K-SAP vs. NO K-SAP</td>
<td>99</td>
</tr>
</tbody>
</table>
1.1 Problem and Motivation

Modern problem solving environments (PSE) are envisioned as collections of cooperating programs, tools, clients, and intelligent agents [156]. These components are integrated into an environment that facilitates user interaction (such a problem statement and solution engineering) and cooperative execution of the components charged with the solution tasks. An example is a system that would help an environmental scientist or a regulator to pose environmental engineering questions (problems), develop, execute and validate solutions, analyze results, and arrive at a decision (e.g., cost-effective emission control strategy). Such a PSE would consist of a management, analysis and computational framework that would be populated with a variety of models and data that describe
the science behind the phenomena, the solutions of interest and the decision rules [46]. Modem PSEs are naturally distributed; new technologies that "guarantee quality of service" are thus especially attractive. One such environment is a computational cloud [1].

A useful abstraction in this context is that of a workflow. It enables us to represent, reason about, program, and manage the complex activities supported by modem PSEs, and the interactions of these activities with information resources, other computational decision support activities of all kinds, and people. A special case are scientific workflows that can include series of structured activities and computations that arise in scientific problem-solving, e.g., studies or experiments. Traditionally, graph-based notations, e.g., generalized activity networks (GAN) and Petri-nets, are used to represent the flow of data, transformations, and machine-to-machine and human-to-machine interactions [10, 46, 52]

In a data-flow abstraction, sequential workflows can be represented by chains of data transformation nodes, or applications, interconnected by data flows. When parallelism and feedback loops are allowed, we talk about networks of communicating data transformation nodes, or application networks.

Traditionally such PSE workflows, would run in relatively private and protected environments (e.g., on personal computers, clusters in data centers). With the advent of cloud computing they have now started migrating in part, or fully, into clouds - often public clouds. With that come some of the concerns that are common in shared environments - security.

1.2 Security

Cybersecurity (or security from now on) is concerned with prevention and detection of unauthorized, malicious or accidental policy-defined misuse of computing infrastructure, data, resources, or applications. While security events are relatively rare events (about 3% of all anomalies detected for software), they can have very high impact and are of special concern. This is particularly true
1.2. SECURITY

in shared environments, such as clouds. Obviously, full cybersecurity encompasses all networking
layers, as well as all middleware layers and all application layers. Security of data is very prominent
component of that. Many security measure and policies are based on the following key, but perhaps
sometimes contradictory, security properties. Those are confidentiality, integrity, and availability
(the CIA triad) [11]

• Confidentiality: Computer related areas assets can only be accessed by authorized parties. A
well known related model is BLP [24] discussed later in the text.

• Integrity: Computer related assets can be modified only by authorized parties. A well known
related model is the Biba model [28], also discussed later in the text.

• Availability: Assets are accessible to authorized parties at appropriate times. Authorized parties
cannot be denied access to the assets [147, 155]. Some common challenges to availability in
information security [105] are (1) Denial of Service; (2) Loss of capabilities due to natural
disasters; and (3) Equipment failures during normal use.

As part of this work we have examined security vulnerabilities and countermeasures in cloud
environments [84, 86]. Security of workflows has also been studied [1, 31, 72, 97, 157] but less so in
the cloud domain. There are a number of papers that address data flow security [57, 102, 131, 163].
Hung and Karlapalem propose an authorization model with a multi-layered (workflow, control and
data) state machine to monitor the flow of authorizations at different layers for a secure workflow
execution [72]. Their model is used to ensure the security properties of integrity, authorization, and
availability. Atluri and Warner discuss the security requirements pertaining to workflow systems
where access control is used to ensure that authorized roles execute tasks within a workflow [15].
Watson discusses a multi-level security model for workflows on a cloud [157]. Using above principles
and lessons learned from investigation of cloud and workflow security issues, we have developed
a workflow security model and a set of tests that we propose to use to gateway flows in and out
of application networks with the intent of increasing application network security. Later in this
dissertation we discuss measures we believe can help secure cloud-based workflows, or verify that
they are secure [23, 85, 86].

1.3 Cloud Computing

Cloud computing has been around for over a decade [1, 13]. In this work we use the NIST definition of
cloud computing [103]. Essential characteristics are (1) On-demand self-service, (2) Broad network
access, (3) Resource pooling, (4) Rapid elasticity, and (5) Measured service. Four deployment models
are (1) Private cloud, (2) Community cloud, (3) Public cloud, and (4) Hybrid cloud. The three service
layers are [1, 13, 103]:

• Software as a Service (SaaS): This is the application layer that most users access remotely.
A SaaS application uses services made available through PaaS, and IaaS, as well as its own,
application specific, functions. At this level, most of the vulnerabilities stem from issues in
the application itself. However, PaaS weaknesses such as those in the underlying operating
system or hypervisor may also manifest.

• Platform as a Service (PaaS): PaaS is the platform on which end-user facing applications are
built upon. The middleware PaaS offers may range from operating systems, to hypervisors, to
programming language support, to different libraries and services, and service construction
tools. PaaS allows for higher-level programming with dramatically reduced complexity; the
consumer does not manage or control the underlying cloud infrastructure including network,
servers, operating systems or storage. Multi-tenant related vulnerabilities are one of major
concerns.

• Infrastructure as a Service (IaaS): IaaS offers physical and virtual resources, interconnection,
and security. Deployment models include private cloud, community cloud, public cloud, and
hybrid cloud. IaaS vulnerabilities are the usual ones for data centers including physical access and firewalls.

There are many cloud service providers. An example of an open-source cloud computing platform specifically designed for education and research is VCL [17, 134]. We used VCL to conduct some of the experiments related to this work reported here.

Figure 1.1 Illustration of a workflow management system
1.4 Application Networks and Workflows

As already mentioned, interacting software applications may form into application networks. Applications communicate with each other by exchanging data. For example, an experimental procedure can become a network composed of many steps, such as preparation, conduct, and analysis of experiments with each step using one or more applications that may run locally or remotely. Scientific workflows are a member of the general category of automated workflows [94]. By integrating many different components such as web services, database server, application servers, and so on [9, 101], into automated flows can simplify implementation, reduce the effort required to create executable models, and automate data manipulation steps. Workflow literature is extensive, and a good classification of workflow models, from serial, to process networks, to discrete is presented as part of Kepler documentation [81]. Figure 1.1 illustrates how a workflow (or application network) management system can be viewed. The graph based representation we call the control plane. It consists of actions (bubbles, processes, transformation nodes) which interact through data flows. A more mathematical view of the workflows can be obtained through Generalized Activity Networks [51, 52]. Each bubble of the directed graph is a process that transforms the data. This can be done through local processing, or by going to external resources. Workflow engine orchestrates, sequences actions and flows (e.g., serial, parallel, process network engine), and selects the type of resources on which actions will run, as well as possibly the physical locations of the resources. All or none of the layers could reside in a cloud. A typical workflow management system should have a control plane that is secure and "understands" the workflow communications, data, and resource needs. Control plane (along with its workflow engine) then communicates and manages step by step execution of the workflow either locally or remotely. Data-flows between resources and the control plane are usually low volume data, metadata, control commands, etc. "Heavy lifting" and high-volume data exchanges are usually relegated to the resource layer. An example of a scientific workflow management system (SWFMS) is Kepler. Kepler and similar systems are now migrating
into clouds [154, 166, 168]. However the security of SWFMS in a cloud is still an open issue [1].

1.5 Approach

We approach the problem of security of workflows/application networks by studying it in the context of Kepler SWMS which we enhanced to include security oriented functions. We discuss control plane abstraction running in a secure environment as a means of implementing appropriate checks on the security properties of the resources and flows. Obviously, the control plane and the workflow engine need to be secure (e.g., in a trusted zone). The data flows among workflow elements and to and from the cloud are used to also verify security properties. As part of this, we propose leveraging provenance collection and interpretation functions available in some SWFMS, such as Kepler [7], Airavata [101], Askalon [54], Kepler [9], Pegasus [43], Taverna [159], Triana [146], and Trident [20] to create security properties checking engine and mechanisms.

1.6 The Structure of Dissertation

Chapter 2 addresses the workflow and application network security model we propose. Chapter 3 and Chapter 4 offer information on vulnerabilities and countermeasures related to cloud SaaS and PaaS layers respectively. Chapter 5 discusses the Kepler implementation of our model. Chapter 6 briefly discusses overhead incurred by our implementation. Finally, Chapter 7 concludes this dissertation and discusses future work.
2.1 Introduction

A workflow is a network of interconnected processing nodes each performing a different function. Figure 2.1 is a data-flow graph representation of a simple workflow. This workflow can also be described by data transformation function $f_s$:

$$\bar{x} \xrightarrow{f_s} \bar{y}$$

where $\bar{x}$ is a vector or a set of inputs into the workflow, and $\bar{y}$ is a vector or a set of outputs from the workflow. Transformation $f_s$ can be decomposed into intermediate step transformations, and their inputs and outputs, spanning many components;
In its simplest form $f_s$ is an application or function chain, i.e., a serial or sequential workflow. In that case equation 2.1 can be expanded into

$$
\bar{x}_1 \xrightarrow{f_1} \bar{y}_1(\geq \bar{x}_2) \xrightarrow{f_2} \bar{y}_2(\geq \bar{x}_{n-1}) \xrightarrow{f_{n-1}} \bar{y}_{n-1}(\geq \bar{x}_n) \xrightarrow{f_n} \bar{y}_n
$$

(2.2)

where $\bar{x}_1$ is the first input vector in the workflow, $\bar{y}_1$ to $\bar{y}_n$ are the outputs from each intermediate transformation $f_i$ for $i = 1 \ldots n$. $\bar{y}_n$ is the final output of this workflow. Each vector/set $\bar{x}_i$ and $\bar{y}_i$ may consist of a number of elements. These elements could be data in various forms, but also parameters describing input/output characteristics such as data velocity, volume, variety (type of variables), and qualifiers such as veracity (trustworthiness of the data and of the input/output channels). Depending on the nature of the transformation functions, their inter-dependencies and the implementation details, parallelism of execution of the functions (actions) and of the flows is a natural extension that in general terms forms a transformation (or application) network. The flows can be discrete, continuous, time-driven, blocking or non-blocking, and so on, and that depends on the processing model chosen.

Figure 2.2 illustrates what a single transformation component may look like. It has a general functional part (which could be as simple as computation of a sine function, or it could be a complex orchestration function for reaching into different cloud resources and validating results), a
communications handler, communications channels from and towards other nodes in the control plane, and/or channels towards outside. In Figure 2.2, input, output and external communication channels (including access to external computational, storage and other resources) are shown as orange boxes. At the control plane (Figures 1.1, 2.1 and 2.3), a component executes computations and data transformations, locally or remotely. Specifically,

- **Process** directs other parts of the component to carry out its function.
- **COMP (Computation)** conducts its own internal operation.
- **COMM (Communication)** communicates with remote resources (i.e., cloud resources and services) regarding any processing and data that may be remote. Communication flows to outside, and management of the control plane flows, is effected by the workflow engine.
- **Input** ($\tilde{x}_i$) indicates input consumed by the component ($f_i$).
2.1. INTRODUCTION

Figure 2.3 Illustration of a workflow management system. Control plane and workflow engine are assumed to be secure.
• **Output** \( \bar{y}_i \) indicates output produced by the component \( f_i \); it could be another new inputs for a subsequent component.

Provided that security of a node (component) is assured in some way at the start of workflow execution, it is assumed that any malfunction (barring hardware or software failures of the platform on which the component is running) will be the result of either internal input flows or external input flows. If we assume that the control plane and its local nodes are secure at the start of the workflow execution, then to keep them secure, assurances are needed that the components remain secure. Such attestation could be direct (actual tests), or it could mean reception of messages and signals from underlying layers (workflow engine and cloud) assuring that the resources are secure and the requested operation has been completed securely.

### 2.2 Elements of Model

The basic principle is that of asserting that for a set of functions operating in the workflow (intra workflow and to the outside), all of their inputs (and outputs) are not unexpected, and that the security properties of the workflow are maintained. Based on elements of the model for secure computer systems [24, 89], let **5-tuple**

\[
WM = \langle F, O, C, SP, SC \rangle
\]

(2.3)

describe the workflow and its security properties

- **\( F \)**: a set of operations (functions, processes, transformations, etc), e.g., \( \{ f_1, f_2, \cdots, f_n \} \)
- **\( O \)**: objects; data, data objects and flows
- **\( C \)**: connectivity matrix for directed graphs describing the workflow, loops are allowed
- **\( SP \)**: security property, e.g., \{ Input, Remote, Data \}
2.3. Securing workflows

The model we use (Figure 2.3) assumes a network of working application nodes operating either locally or in a cloud. They should be controlled and monitored through a secure control plane (and its workflow engine) that keeps a real-time projection of the cloud based and local data-flows, communications and transactions. Operational network nodes (data transformation points) could be in a secure environment themselves, but they may not be. Data flows among the network components and to the outside are monitored and assessed for trustworthiness of the resources, inputs are validated with respect to a known and tested operational profiles (see below), and data integrity is checked. An anomaly may require a decision as to whether the data-flows and transformations/computations should continue, and how that affects confidentiality, integrity and availability expectations and balance. If a run-time failure is declared, a recovery process may be initiated.

As already mentioned in current work we used the Kepler workflow model and implementation [81] as the base-line framework. Kepler allows a variety of data-flow processing models - from fully parallel process networks, to sequential workflow models (see Chapter 5). We have augmented
the standard Kepler provenance collection capabilities with a security analysis package (Chapter 5) intended to help assess security of application network input/output flows.

The basic principle is that of asserting that for a set of functions operating in the workflow (both internal and external flows), all of their inputs (and outputs) are not unexpected, and that the security properties of the workflow are maintained. Therefore, there are three general assumptions:

(a) The control plane (and the platform workflow engine) is secure and has been tested well with respect to the operational profile (expected operating conditions, see below) of the intended workflow.

(b) Intentional corruption of a workflow node occurs only through inputs that are passed to it directly from its predecessor nodes, or through communications with external resources.

(c) Inputs and outputs coming from trusted nodes that operate within control/workflow engine trusted zone are trusted.

This approach, while not full proof, basically states that given a "secure" environment, any attacks (barring insider attacks) can come in only via a limited number of channels: remote access channels (e.g., going to an untrusted URL, or receiving input that may be unexpected or malformed), and data-bases or other data sources that may or may not be in the trusted zone. It is also assumed that any deficiencies in the functional design/implementation of the nodes in the trusted zone that may be triggered by inputs/signals from expected sources will not result in a security breach (but could for example reduce reliability/availability of the system). So long as appropriate countermeasures are set-up at a limited number of known gateways into the secure zone, there is a very high probability it will remain secure.
2.4 Operational Profile

The verification of flows can be done in many ways. The approach we discuss in this work is focused on verification via operational profile (OP) of the workflow. OP is a set of operations and data and their probabilities of occurrence [110, 155]. There may be other channel and data characteristics that could be used in asserting trustworthiness, including data velocity, volume and variety (data types and ranges), encryption and multi-factor authentication. OP is a quantitative representation of how a system will be used. For example, if request for operation A occurs 60 percent of time, B occurs 30 percent of time, and C occurs 10 percent of time, then the "normal range" for requests to these operations is \{(A, 0.6), (B, 0.3), (C, 0.1)\}. Of course, in real life some tolerances need to be built into such range boundaries, and a number of different variables may need to be assessed. It is important to stress that OP description is not limited to input data and operations requests, but must also cover outputs from an application/system (see next paragraph). OP may also vary over time and under different legitimate environmental or use conditions. If a system experiences a different profile (i.e., out of range based on a normal range) a system may have a malfunction by system failure, may have been attacked by a malicious users or software (e.g., computer virus), may have changed environment, etc.

A good example of OP violation is the recent Heartbleed attack [40, 152]. "In SSL, an extension called heartbeat keeps a session alive provided both parties want to keep the session alive even though they currently have no data to exchange. It consists of a payload and a matching response to make sure that the connection is functional. The payload is supposed to be limited to 16KB according to RFC 6520 [50]. However, the vulnerable implementations had a missing bound check on the size of the payload. Since the response is supposed to contain the original payload sent to the server, the server copies this back for the response. Let a small payload, say one byte, be sent, but let the length of the payload be set to 65,535 bytes. With the bug present, the server does not check the size, and copies back into response to client 65,535 bytes. This means the library copies..."
from the memory segments it was not meant to read. This allowed attackers to get passwords and private key information. In a protected workflow, this problem would have been discovered through a simple input/output protocol-aware (application level) channel volume check.

Initially, we will assume that workflow operating under its normal operational profile is more secure than the one receiving or emitting unexpected and/or non-operational profile inputs, output, and metadata. This may not always be case, but it serves to explore the concept behind our model. Security issues are rare events and attack vectors often have combinations of parameters, or signatures that have very low probability of occurring naturally or are prohibited.

The violation of a flow verification rule then implies a possible security issue. Therefore, a flow vector (both input and output) needs to be an element of the component’s operational profile. We assume that for every $i = 1, \ldots, n$

$$x_i \in OP_i \quad (2.4)$$

where $OP_i$ is the "safe" operational profile of function $i (f_i)$, i.e., a set of expected inputs into component $f_i$. This set ($x_i$) consists of both within-workflow inputs, and inputs coming from outside (i.e., COMM channel). It is possible that some special (perhaps untested) combination of allowed input vectors from $OP_i$ can cause a security event. But, also it is assumed that the probability of such event under normal OP is very low. Validation of the inputs, outputs, and communications of workflow elements is recorded in provenance and security analytics modules.

Similarly, we also assume that "output $y_i$" is a member of accepted $OP_i$, i.e., it is an expected output, can be expressed as:

$$y_i \in OP_i \quad (2.5)$$

### 2.5 Verification function

So how do we know what is within an OP? OP is intended to be a quantitative representation of how a system will be used and what to expect out of a system. In theory, all that should have already
been specified, recorded, and available from the software development cycle. In practice, it may be necessary to augment that information through a learning cycle either during testing, or in the field. The ultimate goal is to assess the flows against a "normal" range and/or against the set security characteristics (confidentiality, privacy, availability). In some situations, OP can be easily formulated and constructed, in some cases this is a much more challenging task.

For example, in the case of a square-root (SQRT) function the following implicit equation could be used to check on the returned value of the function inputs and on the integrity of the function outputs [55].

\[
\text{for ( } 0 < q < z) \\
\text{ABS}[\text{SQRT}(q) \times \text{SQRT}(q) - q] < E
\]

(2.6)

where 0 to z defines the range of parameter \( q \) (typically a positive real number), and \( E \) is the permissible error size.

In a more complex case a discrete white list or a black list could be used to allow or block out a suite of permissible URLs (with known parameter values) and this limit the variety and number of external URL requests to a web-site. In general firewalls could be formulated for every layer from networking, to middleware, to application that receives inputs or sends outputs. However, such solutions may result in false positives (and false negatives) and need to be maintained to keep up with user profiles. Alternative solutions may be based on self-adjusting functionally equivalent redundancy - something that a cloud can easily provide (e.g., [152]).

In general, exhaustive description of an OP may be too extensive or too difficult to pin down and we may need to be satisfied by a set of "tests" \( \tau \) against which membership in OP would be assessed. Obviously, probability that we have false alarms increases if \( \tau \) is not identical to OP. For example verification function \( f_v \):

\[
f_v(\bar{x}) = \begin{cases} 
1 & \text{if } \bar{x} \in \tau \\
0 & \text{otherwise}
\end{cases}
\]

(2.7)
where $\tilde{x}_i$ is an input vector/set and $\tau$ is a set of tests which are a verified subset of OP. Set $\tau$ could be generated from OP or it could be an enumeration, or it could be an equation, etc. Attestation of the security of the components implementing a particular function would be needed and could be used to reduce or even skip security tests at the component level. For example, let the component be a function calculating $\sin(x)$ operation by invoking a "sine" service from the cloud. If the cloud can provide an attestation (e.g., certificate) that the resources there are safe and computation ran well, then one could accept the incoming value without further testing for security issues. Correctness may still be an open issue in the context of the application the component is part of.

### 2.6 Security Properties

#### 2.6.1 Threats

Workflow security threats and desired security properties need to be clearly defined [23]. In Chapter 5, we describe a practical framework based on Kepler. In that context we need a provenance (or history) tracking module (which Kepler has), and we implemented a Kepler security management or security analysis package that leverages provenance data and complements that with the ability to check individual input/output flows at the level of the control plane [81, 85].

In order to understand the nature of the threats, and of the appropriate countermeasures, we performed an extensive survey of issues found in SaaS (Chapter 3) and PaaS layers (Chapter 4). We also investigated all Kepler components/actors and directors - a total of 345 (Chapter 5). From that, we developed an understanding of the gating countermeasures that would keep the augmented Kepler framework secure. We identified three categories of countermeasures - input validation, remote access validation, and data integrity assurance.
2.6.2 Input Validation

Input validation detects unauthorized input before it is processed by a component [120]. Input comes from a variety of sources such as end-users, another component, database, and remote resources. For example, if a database is shared with another application, it potentially has security hole, or its data may be corrupted. Even valid resources may include incorrect data [87]. It stands to reason that all inputs should be checked and validated. Input validation could resolve some of known vulnerabilities such as buffer overflow, injection attacks, DoS attack, and de serialization vulnerability. The heartbleed bug turned out to be a missing bound check due to the weak input validation [40]. In general, clear input classes and semantics need to be established because without a clear definition of inputs it may be difficult to resolve anomalies and decide on whether they represent a true threat [32]. We distinguish (1) component Input, (2) component Output (result), (3) Local resources, from a local file system; and (4) Remote resources (e.g., cloud services).

A whitelist model (positive security model)\(^1\) is an example of an input validation specification of the characteristics of inputs that are allowed. A whitelist is a good model when we know exactly what is desired; a blacklist is a good model when we know exactly what we do not want. Another example is definition of patterns through regular expressions. Unfortunately, developing regular expressions can be complicated [120]. Often, it is a combination of complementary countermeasures that is effective.

2.6.3 Remote Access Validation

Remote refers to the need reach beyond the safe and secure core environment of a workflow (application network) to a cloud service, some other external resources, and similar. Remote access validation is needed to protect the internal framework against malicious outside resources. Remote resources are at risk because they may not have been confirmed secure, and whether legitimate or

\(^1\)https://www.owasp.org/index.php/Positive_security_model
not, may have been corrupted. To validate remote resources we check at least two things:

(a) Do we access to the right service address (e.g., URL)?

(b) Do we have the right resources?

To validate if a given address we either need an explicit list of external addresses which the component is allowed to access, or an explicit list (or matching expression) of places the component is not supposed to go. One could also use a trusted reputation service to get an up to date list of trusted services. Furthermore, if the workflow is offering a service on port X, one would also need to vet incoming requests to port X. That may be more difficult unless only a specific set of external addresses is allowed to access the workflow service. In either case address validation is not enough. The actual remote resource needs to be checked for soundness. That can often be done by exchanging keys, or by using an accepted service protocol. To validate if we have the right resources, precondition check should be executed before the workflow accepts the connection. Precondition check is not a component module test, test of the validity of the remote resources. A simple example is a service on port 80. Normally it is expected to be an HTTP service, but it does not have to be. If, instead it is an ssh service protocol mismatch will result in an anomaly and the connection should not proceed. Unfortunately, while necessary, this is not a sufficient condition for maintaining security. A corrupted HTTP resource may pass both the address and service validation test, and then it may attempt to send malformed or specially formulated requests/inputs that would attack the workflow. Therefore, in addition to remote access test, we may also need to validate the input/query stream.

### 2.6.4 Data Integrity

The data that a workflow uses need to be validated. Data may face a broad range of both internal and external threats it its data integrity in the cloud. For example, if we are concerned that a file or data have not changed, we can use hash values for a given data. By comparing hash values (i.e., values before, during and after a workflow execution), compromised data can be detected. However,
other methods are available - encryption, error-correcting codes, including convolutional coding, etc.

In fact, integrity is a basic security property that protects the accuracy and completeness of assets [147]. A well known model in that context is the Biba model. It preserves data integrity by preventing data from a low integrity environment polluting high integrity data [100]. It has two strict integrity policies [28]. Let \( i(z) \) denote integrity of \( z \), then: (1) simple integrity property: subject \( s \) can read object \( o \) only if \( i(s) \leq i(o) \) (Read Up), and (2) integrity *-property: subject \( s \) can write to object \( o \) only if \( i(o) \leq i(s) \) (Write Down). This indicates that a subject's integrity cannot be altered by reading lower integrity information; a subject cannot alter higher integrity information by writing into it.

Confidentiality is another basic security property [57, 102, 131, 163]. Assuring confidentiality assures that information is not made available or disclosed to unauthorized individuals, entities, or processes [147]. Preventing information leakage of sensitive personal data is important (e.g., passwords, keys, and credentials). A well known model called BLP model ensures the confidentiality aspect of access to data. Each object \( (o) \) is assigned to a security level. Each subject \( (s) \) is assigned to a strict level of access that allows it to access all objects with the corresponding security level or below. BLP model has three security properties as follows [24, 102]: (1) "simple security condition property" (Read down), (2) the "*-property" (Write up), and (3) the "discretionary-security property". BLP protect both internal flows and internal processes \( (f_i) \).

This suggests that input data need to be augmented with additional attributes that can indicate its integrity and confidentiality level. In our model we examine all data flows (inputs) and access credentials coming into the workflow, as well as all accesses to the data in the workflow with this in mind.

A security level is assigned to each subject and each object in the system. According to computer systems and countries, the security classification levels vary and the definition of each level could be different. The security level is an element of a hierarchical ordered set. For example, generally, the
hierarchical set (multi-level) consists of Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U), where TS > S > C > U. The security level is associated with the rights between subjects and objects.

### 2.7 Validation process

The basic validation process for our data flow model is illustrated in Figure 2.4. Its implementation prototype in Kepler environment is discussed in Chapter 5.6. The approach we use hinges on (1) an ability to limit attacks to Input, Remote, Output channels (or flows), and (2) validate the flows using operational profile or certification based signals. OP based validation is a statistical approach and may miss some of the attacks. However, where enumeration is possible (e.g., static web sites), this approach can offer high assurances of validity of the flows (e.g., [127]). It is also assumed that workflow components are sound so long as the input flows are limited to operational profile. Other acceptance testing approaches could be used to validate the flows. However, the key element is we do need to always check "external" inputs, and that at some point, if an anomaly is detected a decision needs to be made regarding its significance.
2.7. VALIDATION PROCESS

2.7.1 Adjudicator

If preservation of certain security properties of the workflow is important, then its operational profile needs to contain that information. Detection of an anomaly in an input (e.g., not compatible with OP) triggers invocation of an adjudicator that needs to make a decision regarding the impact of the anomaly on the security properties of the workflow components. Unfortunately, if strictly applied (and provided integrity and confidentiality level of the communicating objects is available) BLP (confidentiality) and Biba (integrity) can be contradictory. BLP is a WURD model (Write Up, Read Down), Biba is RUWD (Read Up, Write Down) [100]. Once an input comes in, the adjudicator should decide which way to go. Many factors should be considered for the decision.

2.7.2 Access Control

Access control limits the activity of legitimate users with the security policy of the organization. A security policy is designed to protect and manage objects (in this case workflow components, and the workflow itself). Access control models such as BLP and Biba deal with information (data) flow, where they focus on confidentiality and deal with some aspects of integrity [132] while availability is a separate property [87]. However (with reference to Figure 2.4) it is possible to make access control decisions that balance confidentiality, integrity, availability and perhaps some other properties. This is further discussed in Section 5.6.
3.1 Introduction

The NIST (National Institute of Standards and Technology) defines three service delivery modes for cloud computing: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS [103]. Building secure cloud systems is the key to their broad acceptance. There are a number of survey papers that address overall security in cloud computing environments [27, 56, 77, 133, 135, 139]. In addition, several organizations such as CSA, MITRE, and OWASP have published lists of vulnerabilities that may occur in cloud computing environments [37, 106, 119]. In recent years, SaaS has become a very significant mode for service delivery. As the demand for SaaS applications increases so do concerns about the security of the data and transactions in that environment. This
concern is not unfounded. Significant security vulnerabilities have been found in SaaS domain. For example, relatively recently (April 2014), the heartbleed bug was discovered in the popular OpenSSL library used in many SaaS services. It turned out to be a missing bound check due to weak input validation \[40\].

This chapter surveys major known SaaS vulnerabilities and the corresponding countermeasures reported since 2009. SaaS can be decomposed into three components (i.e, Application, Network, and Access Control) according to the complemented cloud stack \[56\]. We divide each component into several categorizes based on the correlation of vulnerabilities with countermeasures. This categories cover the space of CWE/SANS top 25 software errors and OWASP top 10 security list \[106, 119\]. We discuss (1) the SaaS security landscape; (2) a classification of survey of vulnerabilities and corresponding countermeasures into twenty categories common across SaaS applications, grouped into three groups; and (3) we briefly discuss SaaS security trends.

The Chapter is organized as follows. Section 3.2, 3.3, and 3.4 address vulnerabilities and corresponding countermeasures for each component, Application, Network, and Access Control, respectively. Finally, Section 3.5 discusses SaaS security trends we found throughout this research.

### 3.2 Application

The application component of SaaS are the applications offered by service providers. Service providers deliver SaaS applications to customers over a network. The customer can access applications via a web browser, thin client, terminal, etc \[56\]. Figure 3.1 illustrates five principal vulnerability/attack categories we use (i.e., injection, xml related, metadata related, man-in-the-browser, and “watering hole”) and the potential countermeasures.

Top categories of vulnerabilities of the application component are injection flaws such as buffer overflow, command injection and SQL injection \[106, 119, 140\]. Another type of injection is a Cross-Site Scripting (XSS) where attackers inject a client-side script into web pages \[106\]. The
3.2. APPLICATION

Simplest way to prevent an injection attack is to keep untrusted data separate from execution codes such as commands and queries. Untrusted data should be separated from the active browser content because working with active content could exploit the interpreter in the browser [119]. Code verification may be necessary if the injected code is written by a client [139]. Input validation is one way of doing that. But it is not a complete defense as some applications require special characters as an input. While there are automated tools for detection of XSS vulnerabilities, it is difficult to detect XSS vulnerabilities completely via automated tools because each application builds output pages differently on different browsers. OWASP suggests that complete coverage should require a combination of manual code review and penetration testing [119].

The vulnerabilities related to Extensible Markup Language (XML) still play a major part in SaaS since many of SaaS applications provide access through web browsers. XML is widely used to support web services (e.g., SOAP, REST, and WSDL) because XML has many advantages such as human readable format, extensibility, platform independence, and compatibility with Java. However,
XML is vulnerable when working with and parsing XML files due to a significant processing overhead. Jensen et al. [77] showed vulnerabilities in web services and discussed countermeasures to resource exhaustion due to XML processing overhead. In this context, a special kind of Denial-of-Service (DoS) attack arises when using namespace [77]. It is not only necessary to apply strict schema validation, but also to restrict resources such as the total buffer size for XML message [77]. JSON (JavaScript Object Notation) can be used as an alternative to XML to mitigate processing overhead that can lead to a DoS attack. XML security can be reinforced based on the WS-Security standard for SOAP and strict rules for DTD (Document Type Definition) and XML Schemas [111]. Data integrity in XML is one of the most critical elements in SaaS applications [139]. WS-Transaction and WS-Reliability standards are used to manage data integrity at API (Application Program Interfaces) level via HTTP connection which does not support transactions or guaranteed delivery [139]. However, not many service providers have implemented these standards [139].

SOAP (Simple Object Access Protocol) based on XML is a lightweight protocol used to exchange services related structured information. SOAP data is vulnerable to a variety of MITM (Man-In-The-Middle) attacks, such as interception, manipulation, and transmission, so WS-Security using XML Signature and XML Encryption is used to protect against such attacks. However, naive use of XML Signature is vulnerable to wrapping attacks that add extra elements onto SOAP messages. Although XML Signature is designed to facilitate data integrity protection and origin authentication for a variety of document types, it may still lead to security problems unless web service developers are aware of some subtler properties of XML Signature. XML documents containing XML signatures are processed with two steps: signature validation and function invocation (i.e., business logic). An attacker can alter the message structure by injecting forged elements, which still validates the XML signature. Unfortunately, application logic may follow different parts of the message which may not be able to validate the XML signature. The countermeasure is to forward only signed elements that guarantee each module accesses the same elements used in both signature validation and application logic [138]. An attacker can try to guess the omitted operations and call them by
scanning WSDL (Web Services Description Language) text because WSDL advertises a service's operations (e.g., parameters, data types, and network bindings) to either local network or the outer network, called internal operations and external operations, respectively [77]. If the web service is created using common web service framework tools, the generated WSDL contains all operations in one place. In this case, an external client gains knowledge of the internal operations and can invoke them. Deploying the internal and external operations to separate web services is to prevent WSDL scanning attacks [77]. Web-Service-aware XML firewall can be used to protect WSDL against scanning attacks if separating web services is not applicable [77].

A web service client collects metadata information (e.g., message format, network location, security requirement, etc.) from the metadata documents provided by the web service server. This metadata is usually distributed via insecure communication protocol (i.e., HTTP) that leads to MITM attack such as data modification. To avoid such metadata spoofing, all metadata documents should be checked thoroughly by for example verifying a hash value of the metadata description [76].

Man-In-The-Browser (MITB) attacks occur by means of trojan horses that tamper with the contents of web pages and transactions in web browsers [124]. Neither the user nor the server can detect any change of data before it is sent to the server because the malware acts between them by delivering fallacious information to both parties [124]. Although MITB attacks are similar to MITM attacks, the countermeasures against MITM attacks are often not applicable to MITB attacks because MITB attacks act at the application layer while the countermeasures, such as authentication mechanism and secure communication channels (i.e., HTTPS), operate below the application layer [124]. Rauti and Leppänen [124] propose mitigation of an MITB attack in the Ajax application's JavaScript code by (1) encrypting data; (2) using random function name; and (3) setting up time limit to decipher.

A “watering hole” attack exploits vulnerable websites which victims visit frequently [142]. The attacker compromises the website by injecting JavaScript or HTML to redirect victims to a separate site which includes malicious code. Several zero-day exploits used in watering hole traps have been
discovered in several applications such as Adobe Flash Player, Microsoft Internet Explorer (IE), and Microsoft XML Core Services [143]. The best way to prevent this attack is to apply all relevant patches as soon as they are available [143]. Symantec [144] offers the comprehensive practice guidelines, which can be applied to SaaS applications extensively.

3.3 Network

![Security vulnerability categories and countermeasures of the Network Component](image)

Figure 3.2 Security vulnerability categories and countermeasures of the Network Component

The network component of SaaS should provide secure connections to protect information from being viewed or modified by a third party. Traditional network security issues lurk on insecure networks such as MITM attack, IP spoofing, port scanning, packet sniffing, etc [139]. Secure Socket Layer (SSL) /Transport Layer Security (TLS) is widely used to secure data transfers by encryption against traditional network security issues.

Unfortunately, researchers have discovered serious vulnerabilities in SSL/TLS mechanisms. Rivest Cipher 4 (RC4) encryption in the cipher suites for SSL/TLS has flaws that an attacker can use to recover a limited amount of plaintext from a TLS connection [5, 6]. The following countermeasures are recommended [6]: (1) using other cipher suites such as CBC-mode and AEAD; (2) patching TLS's
use of RC4 by discarding the first output bytes of the RC4 keystream; (3) modifying browser behavior by using HTTP GET requests; and (4) limiting the lifetime of cookies or the number of times users can be sent from the browser. However, CBC-mode encryption also has flaws that a MITM attacker can exploit [5]. The countermeasures include (1) adding random time delays to CBC-mode decryption processing; (2) using other cipher suites such as RC4 and AEAD; and (3) ensuring uniform processing time for all ciphertexts of a given size.

Pseudo Random Number Generator (PRNG) is the elementary and critical component for generating a key pair for the public key cryptography. Bad randomness properties of PRNG may cause vulnerability in RSA (Rivest, Shamir, Adleman) and DSA (Digital Signature Algorithm) used in TLS cipher suite because an attacker might be able to predict a public key [25, 69]. Bellare et al. [25] show that they mitigate randomness issues through hedged public key encryption. Heninger et al. [69] suggest defensive strategies against bad randomness for several important groups of stakeholders. For example, (1) application developers should generate keys when they have sufficient entropy and (2) end users should regenerate default or automatically generated keys. They have created an online service (https://factorable.net) that allows user to check their SSH host keys or TLS certificates against their dataset.

As the demand that SaaS applications manage sensitive data grows, communications with applications are transitioning from HTTP to HTTPS. Also, wariness about cookies is on the rise. Gollman [60] addresses cookie poisoning. Cookie containing the user's credential information can be altered by unauthorized users [27, 60, 121]. To prevent this, a message authentication code can be used to protect the integrity of a message, identity of the originator, and non-repudiation of origin. Regular cookie cleanup is recommended [27]. Dacosta et al. [41] proposed one-time cookies (OTC) to prevent session hijacking attacks with stateless authentication tokens. OTC is implemented for HTTP cookies as an alternative to HTTPS performance overhead. OTC can be combined with HTTPS to improve the security of web applications.
3.4 Access Control

Access control component deals with resources access in SaaS. It consists of three parts: Authentication, Authorization, and Identity & Access Management (IAM).

3.4.1 Authentication

Authentication vulnerabilities (e.g., missing or broken authentication and session management) are one of the high security risks [106, 119]. Although developers build custom authentication and session management schemes, building these schemes correctly is hard because each implementation is unique [119]. For careful and proper use of custom or off the shelf authentication session management mechanisms, OWASP [119] recommends several: (1) password strength; (2) secure transition for protecting credentials and session ID; and (3) trust relationships between components.
Ur et al. [150] found that slightly less stringent password meters may be better because very stringent meters can lead users to abandon the task of creating a strong password.

Hart [68] mentions that two-factor authentication (2FA) is the simplest and most effective method of reinforcing insufficient authentication methodology such as archaic static password and a one-tier log-in. In addition, Dinesha et al. [47] propose multi-level authentication technique because using combinations with weak authentication methods reduces the probability of breaking a password by the orders of magnitude. With the advent of BYOD (Bring Your Own Device), Google introduced 2-Step Verification (2SV) by adding an extra layer of security for logging into Google accounts [64]. Whenever the user signs in to Google, a user will enter his username and password and then the user will be asked for a code that will be sent to himself or herself via text, voice call, or Google’s mobile app. This means that an attacker can not access a legitimate user’s Google account unless an attacker has a legitimate user’s mobile device [64]. Like strong password meters, this can discourage users.

Authentication methods have failure modes that allow resetting the password of an account’s owner [56]. Security question and answers (Q&A) are commonly used to verify account’s owner [66]. However, Grosse and Upadhyay [66] suggest not using the security Q&A approach because there is a finite number of answers to the questions and many would misplace their answers. Authentication mechanism may use an account lockout policy to protect an user account and prevent DoS attacks [65]. The server is to lock out accounts that have received several unsuccessful authentication attempts in quick succession. However, this protection mechanism can be abused by an attacker by deliberately sending wrong password until the target account locks [118]. CAPTCHA (Completely Automated Public Turing Test To Tell Computers and Humans Apart) can be an effective (if not always efficient) way of protection against automated attacks [149]. Another countermeasure is to set the account lockout policy based on comparison of risks between brute force attacks and lockout attacks [148].
3.4. ACCESS CONTROL

3.4.2 Authorization

Authorization is the mechanism of specifying access rights to data resources. The insufficient authorization (e.g., missing authorization and incorrect authorization) is dangerous [106]. Missing authorization processing can lead to the root cause of URL-guessing attacks [65]. The combination of RBAC (Role-Based Access Control) and ABAC (Attribute-Based Access Control) can provide fine-grained authorization [35]. In addition, authorization frameworks such as JAAS and the OWASP ESAPI Access Control feature are recommended [106]. Sometimes, function level protection is managed via configuration which may be misconfigured. Such flaws allow an attacker, who is otherwise an authorized system user, to access a privileged function by simply altering the URL or a parameter [119]. Thus, access control checks using authorization frameworks should be made before a request to sensitive data is granted [116]. Due to the growth of cloud computing, a need for services which integrate heterogeneous data and applications from multiple sources has emerged [56, 136]. Although centralized access control can have benefits, it may not be possible for heterogenous systems [34, 56]. Chow et al. [34] propose information-centric security where data used by system and applications are encrypted and packaged with a usage policy regardless of environment.

3.4.3 Identity and Access Management

Identity and Access Management (IAM) deals with identifying individuals in a system and controlling their access to the resources in that system. A SaaS provider should offer the complete stack of IAM services. As SaaS providers extend their services (e.g., Google Apps including Gmail, Google Calendar, Docs, Driver and so on), the complexity of IAM and the pressures of risk and compliance demand a new approach to controlling and monitoring access to data and applications. To help protect extended enterprise, effective IAM solutions should help centralize access policies and controls via (for example) Single Sign-On (SSO) [74]. For instance, the vulnerability of Google Apps found by Armando et al. is a flaw in IAM synchronization leakage [12, 139]. Moreover,
synchronizing enterprises that users are able to access via external cloud services is a challenging issue due to complex hierarchies. This calls for federated Identity Management (IdM) to reflect the dynamics of cloud computing and its rapidly evolving threat environment [135, 139]. Ahmed et al. [3] summarize ideal IAM functions based on the report from International Data Corporation (IDC) as follows: (1) privileged user control; (2) access management/SSO; (3) user authentication/federation; (4) identity management and role management; (5) data loss protection/prevention; and (6) log management. CSA [38] addresses major IAM functions in the cloud as follows: (1) identity provisioning/deprovisioning; (2) authentication & federation; (3) authorization & user profile management; and (4) support for compliance. Password-based authentication has an inherent limitation [145]. User-centric identity management (IdM) approaches allow users to protect their private and critical identity attributes by controlling their own digital information. Ahn et al. [4] propose a category-based privacy preference approach to enhance the privacy of user-centric IdM systems against the existing IdM system that does not consider privacy issues in depth. The research of building robust IAM has gained much attention recently in the context of Bring Your Own Identity (BYOI) and Bring Your Own Device (BYOD) [114].

3.5 Some Trends

We find two major trends in the context of SaaS security.

First, the Internet is in transition from HTTP to HTTPS as use of sensitive data increases. While deploying HTTPS can be challenging due to performance, Google for example convincingly demonstrated that SSL/TLS is not computationally that expensive [130]. Compared with HTTP, the CPU overhead of HTTPS is less than 1%, less than 10KB of memory per connection and less than 2% of network overhead [130]. However, it should be noted that HTTPS protocol must be applied from the very first connection establishment until the termination to guarantee secure communication. For example, if a website uses HTTPS and then the rest of the session is maintained over HTTP or vice
versa, it does not guarantee security at all because the session cookie can be sniffed in plaintext [56]. Thus, the transition to HTTPS can proliferate by the growth of integration between social networking websites and enterprise websites.

Second, BYOI (Bring Your Own Identity) is the current trend after BYOD (Bring Your Own Device). An increasing BYOD trend has incurred identity fatigue (i.e., users have too many online ID to manage), which has created BYOI that gives users the ability to use one of their existing social media IDs to access enterprise services [71]. Because BYOI solutions are relied upon for securing access to enterprise services, fundamental security issues such as availability, reliability and trust should be considered when such solutions are employed. Thus, BYOI research should be actively explored and studied.
4.1 Introduction

Platform as a Service (PaaS) allows cloud developers and providers of higher-level services to build and deploy applications by hiding complexity of lower level functions and services. Figure 4.1 illustrates cloud service models with PaaS features based on Fernandes et al. [56].

PaaS enables lower costs and higher computing efficiency by enabling better utilization, often based on multi-tenancy, through virtualization technology. There are many PaaS providers (commercial, academic, open source, proprietary, etc.) that offer different types and ranges of features, tools and services. For example, Amazon web services, Google App Engine, LongJump, IBM Bluemix, Salesforce, etc. As long as the products (i.e., applications) developed on top of PaaS are connected
via the Internet, security should be considered for both PaaS vendors and clients. Furthermore, multi-tenancy is the key source of security concerns when multiple PaaS services reside on a single physical server. To achieve safe (or secure) multi-tenancy, PaaS solutions must have a way to isolate tenants from each other. There are a number of survey papers that address overall security in cloud computing environments [49, 56, 123, 135, 139, 161].

This chapter discusses major PaaS security and privacy requirements, and vulnerabilities and the corresponding countermeasures reported since 2009. We consider three potential sources of issues: (1) Software Platform and Isolation; (2) Virtualization; and (3) Data Security & Integrity.

In practice, many PaaS vulnerabilities may overlap with SaaS or IaaS vulnerabilities since they are frequently intercorrelated. Of course, many general vulnerability categories, such as the CWE/SANS Top 25 [106], are present in the PaaS layer as well as in the SaaS layer [86].

The chapter is organized as follows. Section 4.2 address background of cloud computing. Section 4.3, Section 4.4, and Section 4.5 discuss principal categories of vulnerabilities and the corresponding countermeasures. Section 4.6 summarizes PaaS security trends we observed.
4.2 Background

Cloud computing offers services over the Internet. We define the boundary among SaaS, PaaS, and IaaS is using NIST (National Institute of Standards and Technology) cloud model and its service layers as well as essential characteristics and deployment models [103].

PaaS illustration shown in Figure 4.1 includes execution environment (e.g., GAE, .NET, and JVM) and Programming Environment (e.g., APIs, IDEs, and Django). For example, GAE (Google App Engine) runtime environment supports apps written in a variety of programming languages, such as Java, Python, PHP, and Go [63]. GAE includes several features that make it easy to build and deploy an application: (1) Persistent storage with queries, sorting and transactions; (2) Automatic scaling and load balancing; (3) Asynchronous task queues for performing work outside the scope of a request; (4) Scheduled tasks for triggering events at specified times or regular intervals; and (5) Integration with other Google cloud services and APIs, such as computing, storage, networking, big data, services, and management. GAE also provides development environments, SDK (Software Development Kits).

There are many PaaS tools and environments. Their goal is to quickly and efficiently design and deploy applications, and have them function reliably. The important thing to remember is that security flaws that may exist in the PaaS layer can propagate to the SaaS layer. A classical example may be vulnerabilities that may occur in the hypervisor, or in the security tools supplied to applications, such as SSL vulnerabilities (e.g., Heartbleed vulnerability).

An example of an open-source cloud computing platform specifically designed for education and research is VCL [17, 134]. VCL supports IaaS, Paas and SaaS. VCL environments can range from a simple software application to High-performance computing (HPC) resources. From the

---

1 https://cloud.google.com/products/
2 http://heartbleed.com
### Table 4.1 An overview of major PaaS security concerns

<table>
<thead>
<tr>
<th>Category</th>
<th>Vulnerability</th>
<th>Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Platform</td>
<td>isolation for multi-tenancy</td>
<td>isolation at each layer, container [129]</td>
</tr>
<tr>
<td></td>
<td>integrity of cloud services</td>
<td>integrated platform, architecture [98]</td>
</tr>
<tr>
<td></td>
<td>cloud architecture</td>
<td>specific requirements [45, 96]</td>
</tr>
<tr>
<td></td>
<td>data sharing in isolation</td>
<td>IFC [122]</td>
</tr>
<tr>
<td>Virtualization</td>
<td>side-channel attacks</td>
<td>obfuscating internal structures [125], timing channel control [18], cached-based method [165]</td>
</tr>
<tr>
<td></td>
<td>CPU-based covert channel</td>
<td>CCCV [112]</td>
</tr>
<tr>
<td></td>
<td>covert channel in micro-architecture</td>
<td>small allocation of resources, physical and functional hierarchies [30]</td>
</tr>
<tr>
<td></td>
<td>virtualization-based cloud computing</td>
<td>monitoring system [92]</td>
</tr>
<tr>
<td></td>
<td>managing VM images</td>
<td>imaging management systems [158]</td>
</tr>
<tr>
<td></td>
<td>sensitive information</td>
<td>architecture change [29]</td>
</tr>
<tr>
<td></td>
<td>memory disclosure attack on memory duplication</td>
<td>read-only page flag [141]</td>
</tr>
<tr>
<td></td>
<td>limitation of random number generator</td>
<td>external sources [126]</td>
</tr>
<tr>
<td></td>
<td>virtual network</td>
<td>virtual network model [160]</td>
</tr>
<tr>
<td>Data</td>
<td>sensitive data</td>
<td>additional security check, storing metadata information [139]</td>
</tr>
<tr>
<td></td>
<td>data loss and leakage</td>
<td>security components (e.g., identity management, log service, access control), secure block storage [45, 151]</td>
</tr>
<tr>
<td></td>
<td>data integrity</td>
<td>MAC with Merkle tree-based structure [78]</td>
</tr>
<tr>
<td></td>
<td>data relocation</td>
<td>constraint-based data geolocation [61]</td>
</tr>
<tr>
<td></td>
<td>big data processing</td>
<td>efficient security mechanism (HLA, E-PDP) [91]</td>
</tr>
</tbody>
</table>
point of view of security it offers (1) Variety of authentication options (e.g., LDAP, Shibboleth); (2) High security and isolation (e.g., IP-lock, local firewalls, point-to-point VLANs and VPNs, one-time passwords, feedback confirmation, timeout, traffic monitoring); (3) Sophisticated resource access and mapping privilege tree; and (4) Real-time monitoring of reliability and security. We are in the process of developing an NC-wide university data science and analytics cloud environment based on VCL that uses iRODS\(^3\) as the data-bus. VCL also interoperates with commercial cloud offerings - e.g., Amazon and IBM. This is where interesting security and privacy interactions occur. For example, mapping and matching of security properties available in one cloud onto those available in another cloud.

When considering PaaS security, we believe the problem is driven by virtualization (and utilization) encouraged multi-tenancy with the primary problem prevention tool being appropriate isolation of users, resources, and data. Table 4.1 provides an overview.

### 4.3 Software Platform and Isolation

PaaS elements range from Operating Systems (OS), to Hypervisors and Virtual Platforms (VP) with Java and .NET. Unfortunately, such platforms may not offer fully secured hosting environment due to the limitation of isolation mechanism. Being applications themselves, PaaS elements may also harbour all the usual vulnerabilities that SaaS layer elements do \[86\].

Rodero-Merino et al. discuss security vulnerability of clouds where tenants share PaaS platform services [129] and host multiple user applications on the same resources. To achieve safe multi-tenancy, each application should run in an isolated environment. They discuss the limitation of the OS as a hosting environments. For example, PaaS providers may prefer to offer simplified abstractions that ease the development tasks. However, OS may not support a set of applications that could be run in the cloud. Instead, each application could use containers that encompass the resources

\[^{3}\text{http://irods.org}\]
associated with a particular task. They note three isolation options: (1) Isolation at OS level; (2) Isolation by Standard Java Security; and (3) Isolation at VM level. They summarize the main open security issues at different layers (i.e., OS, Java VM, and Container) of a Java PaaS platform and the security features for different VPs. They expect that container technologies could be widely used to run user components. In fact, Docker\(^4\) is such an open platform released in March 2013 that is growing in popularity (see Google trends\(^5\)). It uses resource isolation features of the Linux kernel and provides lightweight containers to run processes in isolation. While Docker solves a number of issues, Docker solution is not vulnerability free [19, 48, 70].

Mackay et al. proposed an integrated platform to reinforce the integrity and security of cloud services [98]. They present key features of the proposed architecture: (1) Service planning: SLA (Service Level Agreement) should be negotiated; (2) End-to-end security: encryption techniques and authentication mechanisms; and (3) Monitoring and enforcement: Performance & Stability metering and IDS (Intrusion Detection System).

Deng et al. address requirements in the context of healthcare applications [45]. They identified core requirements for building trusted cloud architectures using Bessani’s work [26]: (1) Separation of duties; (2) Isolation; (3) Verifiability of trustworthiness; and (4) Integration into subscriber’s IT infrastructure. Based on such core requirements, Deng et al. propose a design of a trustworthy healthcare platform for PaaS by providing several security functionalities including identity management, authorization, and log services, access control and automatic audit tool. Their platform is in compliance with EU data protection and privacy legislation.

Luo et al. present a virtualization security framework based on the requirements and solutions [96]. Virtual security framework could have two modules such as virtual system security with three layers (e.g., Physical resources, VMM (Virtual Machine Manager), and VM) and virtualization security management. They address virtual system security with four parts: (1) VM system security

\(^4\)http://www.docker.com

\(^5\)http://www.google.com/trends/explore#q=docker
architecture: They show three VM security system architectures by comparing each structure in terms of efficiency and flexibility; (2) Access control: They show an access control framework about how resources flow and communicates between VMs and VMM based on access control policy; (3) Virtual Firewall (VF); and (4) vIDS/vIPS: vIDS (Virtual Intrusion Detection System) / vIPS (Virtual Intrusion Prevention System). The virtualization security management could be conducted by either VMM or security control layer, which has four roles: patch management, VM migration management, VM image management, and audit.

Pasquier et al. present how data across applications can be shared in cloud using isolation mechanisms [122]. They present Information Flow Control (IFC) as a flexible option; IFC is an approach to security that allows application writers to control how data flows between the pieces of an application and the outside world [88]. They use tags and labels for secrecy (i.e., confidentiality) and integrity that regulate flows throughout the system; they use BLP model for secrecy and Biba model for integrity [24, 28].

One thing to keep in mind is that if real isolation is desired, then the user needs to "own" the whole stack and so multi-tenancy in any form is probably not a good idea. In that case bare machine loads or sole-use of a hypervisor may be the only safe solutions. Of course, this may come at the expense of utilization efficiency of computational and storage resources.

### 4.4 Virtualization

Virtualization is a major component of cloud computing as we know it today. The growth of cloud computing would not have been possible without supporting virtualization technologies; the use of virtualization enables sharing of resources (and through that it encourages multi-tenancy). This brings the challenges of multi-tenant vulnerabilities [162].

Ristenpart et al. discuss multi-tenant vulnerabilities [125]. They show that adversaries may
be able to identify internal cloud structure. Based on this, a comprised VM is able to mount near a target VMs as a co-resident. This approach can lead to cross-VM side channel attacks due to the sharing of physical resources (e.g., via CPU’s data caches). They suggest several ways to avoid such side-channel attacks. First, cloud providers may obfuscate both the internal structure of their services and the placement policy in order to complicate adversary’s attempts to place a VM on the same physical machine as its target. Second, they enumerate many countermeasures against side-channel attacks. But, they also claim that such countermeasures have drawbacks; these solutions are either impractical due to high overhead and nonstandard hardware, application-specificity, or an insufficiency that prevents full mitigation of the risk. Avoiding co-residence would be an ultimate countermeasure against cross-VM attacks. Finally, they claim that the best solution is simply to expose the risk and have the user decide.

Okamura and Oyama evaluate the threat of CPU-based covert channels between VMs on the XEN hypervisor where different VMs share a physical CPU (e.g., a single-core uniprocessor) [112]. They have developed a system called CCCV (Covert Channels using CPU loads between Virtual machines) that creates a covert channel and communicates data secretly; this enables processes in DomU virtual machines to communicate with each other using a physical CPU as the communication medium. CCCV consists of a sender process and a receiver process. A sender process runs in a domain from which a malicious program sends secrets (e.g., a credit card number and a password). A receiver process runs in a domain in which a peer of the malicious program receives the secret. The communication failure happens when the weights of the credit scheduler for the sender and receiver differ. However, they show that success of communication depends on whether the CPU loads of processes other than the sender and receiver vary significantly during communication.

To avoid timing side-channel attacks, Aviram et al. propose a timing channel control based on provider-enforced deterministic execution by collecting all internal timing channels into a single controllable channel at the cloud’s border, rather than eliminating timing channels within a shared cloud domain [18]. The cloud provider assigns each job onto shared hardware within the cloud with
4.4. VIRTUALIZATION

statistical multiplexing and returns the job's output, which depends only on explicit inputs, not on timing of operations within the cloud. Zhang et al. detail the construction of an access-driven side-channel attack, where a malicious virtual machine (VM) extracts fine-grained information (e.g., private ElGamal decryption key) from a co-resident victim VM running Gnu Privacy Guard (GnuPG) [164]. They introduce numerous sources of channel noise and develop a sequence of mechanisms to refine a support vector machine (SVM) outputs to reduce such sources of noise. They discuss several countermeasures with benefits and downsides, such as avoiding co-residency, side-channel resistant algorithms, and modifying CPU scheduling for XEN credit scheduler. Zhang et al. conducted cache-based side-channel attacks in commercial cloud, PaaS that extract sensitive application data (e.g., the number of items in a shopping cart) [165]. The attacker instance in their framework can trace a victim's execution paths inside shared executables; it should monitor over time to trace the victim's execution path in the CFG (Control Flow Graph). They describe that the existing countermeasures to cache-based side channels may not be applicable to their attacks. Instead, they introduce several countermeasures: (1) disabling the clflush instruction; (2) increasing background noise with more applications; and (3) disallowing resource sharing of memory pages that serve as flush-reload attack vectors.

Caron et al. state that virtualization is the current trend in the cloud and requires security and performance isolation between users and VMs [30]. They describe several vulnerabilities and countermeasures for covert channels in micro-architecture components shared between VMs. Lack of isolation of micro-architectural components may lead to creation of covert channels between VMs, on the other hand, mechanisms with strong isolation lead to a large overhead. They claim that it is possible to reduce covert channel through smart allocation of resources for strong isolation while keeping a good consolidation of the platform. They propose a model that has the physical (i.e., geographical) and functional hierarchies. A functional hierarchy can divide a platform into available zone; it avoids sharing both hardware components and the same component backup. They also show the performance based on the relationship between isolation and consolidation. The
4.4. VIRTUALIZATION

performance depends on the physical isolation of workloads; the contention on memory channels or processor caches on shared physical machine is a major cause of the decrease in performance.

Lombardi and Pietro show how virtualization can increase the security of cloud computing [93]. They propose a monitoring system called Advanced Cloud Protection System (ACPS) by extending the KVMsec [92]. ACPS monitors the integrity of key components such as key kernel and middleware by performing a checksum of such components; it checks behavior and integrity of cloud components via logging and periodic checksum verification of executable files and libraries. Also, they show that overhead due to the additional integrity checks is relatively small.

Wei et al. explore the problem of securely managing VM images [158]. They propose an imaging management system that controls access to VM images by granting access permission. Image filters detect malicious images and remove user-specific sensitive content from the images. These approach, conducted as the general countermeasures, can be used on any cloud, for example, Amazon Elastic Computer Cloud (EC2), to protect sensitive information. In practice, Bugiel et al. could extract highly sensitive information (e.g., passwords, keys, and credentials) from publicly available Amazon Machine Images (AMIs) using an automated tool they developed [29]; the tool discovers the sensitive information (i.e., private keys and credentials) usually in the home directory of users, the home directory of a superuser (i.e., root), and common locations for programs and configuration files. They suggest some countermeasures that would require fundamental changes to the architecture by integrating Amazon specific services into the EC2 Cloud App Store.

Suzaki et al. describe memory disclosure attack on memory deduplication (i.e., COW (Copy-On-Write)) vulnerability on VMs [141]. They show that an attacker can detect the existence of specific running applications and a downloaded file on another VM by using the time difference in write access time between deduplicated and non-deduplicated page because deduplication process takes longer than non-deduplication process. Candidate pages for deduplication are examined during a certain amount of time in order to check if they are identical. A countermeasure to this attack could be read-only pages for memory deduplication since such pages do not need to be re-written.
4.5. DATA SECURITY & INTEGRITY

Ristenpart et al. investigate the failure of random number generators (RNGs) in VM due to reuse of VM snapshots [126]. Because of the limitation of RNSs in VM, they suggest guest RNS services with hardware-based RNGs or other external sources; they have developed a general framework for hedging cryptographic operations.

Wu et al. present a novel virtual network model against the break of isolation [160]. Most hypervisors offer virtual network to connect VMs by using bridge and route while remaining isolation among VMs. However, isolation is easy to be broken in the current virtual network because all VMs share the bridge and route either working as a virtual hub or virtual switch. They suggest that a virtual network model should have three layers: (1) Routing: a set of unique static IDs can be assigned to a shared network by the administrator; (2) Firewall: this layer is to prevent both spoofing attacks and modification of routing table from any packet (i.e., ARPing); and (3) Shared network layer: each VM has a set of subnets.

4.5 Data Security & Integrity

Protecting data and maintaining data integrity is important for all cloud service delivery models. PaaS is the key player in providing end-to-end data security and integrity in multi-tenant situations. The PaaS framework needs to either directly or indirectly support four types of data interaction modes: (1) Total Isolation, (2) Computations-to-Data, (3) Data-to-Computations, and (4) Open Data. In all cases encryption at rest and encryption in motion must be an option, as should be the ability to compartmentalize and isolate access and scope individual users see and access.

Total Isolation mode usually applies to very confidential data category. It may pertain to proprietary data and/or federally regulated data that should be only accessible and shared in a very isolated environment. Private clouds are probably the only suitable environment for that, and even then one
5. DATA SECURITY & INTEGRITY

may want to consider more restrictive physical solutions - e.g., a SCIF (Sensitive Compartmented Information Facility).

Computations-to-Data category may also involve confidential information, often very large amounts of data, that would be available only to authorized and authenticated individuals and devices. Data sets of that type may be in the terabyte and even petabyte range, and may not be easily moved around. On the other hand, there may be situations where we may not want to move the data in this category out of the original storage even if the size is not an issue. For example, local laws or policies may not allow that. Therefore, mechanisms need to exist whereby computations using the data (e.g., analytics) are performed close to the data, i.e., we move computations to the data. PaaS support would typically be needed for this type of operations.

Data-to-Computations category is more along the lines that we often see today. Data in this category can typically be downloaded or moved to other clouds or locations such as end-user desktops. For example, relatively low volume data visualizations may fall into this category, as may relatively small data sets that do not take long to download, etc. This data may also be restricted to authorized and authenticated users.

Open Data are made available to a broader public and may not in fact require any identification of the end user. If the size is an issue, then security-aware computations-to-data model of access may be used, if not, data-to-computations may be sufficient.

However, it is important to note that in order for cloud services, and PaaS layer, to be reliable, trustworthy, secure and safe, the PaaS framework needs to collect provenance information about the users, data sources, transactions on the data, users, and so on. This becomes particularly important if data may have export/import or confidentiality restrictions. Hence, PaaS services may need to worry about compliance issues (including geolocation and privacy). For example, some or all of the data may need to be collected over encrypted channels, and in some cases encryption at rest will be a must. It is also important that the data and interfaces to the data and the framework platforms are usability aware (since that may have human element security implications).
Subashini and Kavitha detail data security for overall cloud service delivery models [139]. They indicate that additional security checks should be applied to the sensitive data to ensure data security and prevent breaches; a PaaS provider has to offer strong assurances that the data remains inaccessible between applications. They claim that a framework should have a functionality of storing meta-data information in different locations which would make information unavailable if a malicious user tries to recover it. Vaquero and Rodero-Merion address that the threats of data loss and leakage are increasing in the cloud due to the architectural or operational characteristics of cloud environments (e.g., insecure APIs, shared environment, etc.) [151]. Deng et al. employ various techniques to protect data in their platform [45]. They show how Health Trusted PaaS should manage sensitive data (e.g., a patient’s data) with security components such as identity management, log service and access control. They introduce secure block storage that provides version control for encrypted data chunks to prevent replay attacks.

Juels and Oprea describe new techniques for data integrity and freshness verification that secure cloud data [78]. For data integrity, they use an authentication scheme by computing Message Authentication Codes (MAC) for each data blocks with 4KB. For freshness verification, they use a counter or version number with each file block which also must be authenticated. They claim that their authentication mechanism is practical and scalable by reducing the storage overhead due to a Merkle tree-based structure. They introduce their auditing framework that offers tenants visibility into the correct operation of the cloud. They mention several open problems as follows: (1) performing computations over tenant’s encrypted data; (2) ensuring tenant isolation; and (3) geolocation of data. Geolocation of data is of particular commercial interest recently because of regulatory compliance and many laws. For example, providers should store customer data within national boundaries. Gondree and Peterson propose a framework of using constraint-based data geolocation [61]. Their framework could bind data to a location with assurance in that it can protect unexpected cloud data relocation. Geographic region options may be able to resolve various concerns such as performance of web services, regulatory compliance of sensitive data, and the delay
An interest in ‘cloud and big data’ grows, so does importance of data protection and trustworthiness. Lie et al. analyze security and privacy in big data application and cloud computing [91]. They propose a lifecycle of integrity verification over big data in cloud computing. As the dataset in big data applications are very large and require heavy-scale processing, the security approaches related to big data and cloud need efficiency. They discuss several effective and efficient security mechanism for integrity verification as well as standard schemes. For example, HLA (Homomorphic Linear Authenticator) or HVT (Homomorphic Verifiable Tag) are technologies that use verification metadata based on homomorphic signatures to authenticate. This approach has evolved from standard digital schemes (RSA and BLS). E-PDP (Efficiency Provable Data Possession) with sampling can achieve better efficiency than S-PDP (Secure Provable Data Possession) [14, 91].

4.6 Some Trends

We recognize three core elements of PaaS services (Figure 4.2). Multi-tenancy is the most popular paradigm for achieving higher resource utilization and economies of scale [67]. Isolation technologies should enable better security in multi-tenancy situations [53]. PaaS services can be co-located through multiple VMs [75] and physical resources can be shared transparently across VMs to the benefit of multiple users.

We observe three major trends in PaaS security. First, multi-tenancy is here to stay, and so are multi-tenancy related security concerns [79]. As discussed in Section 4.3, there are various isolation mechanisms that can help mitigate multi-tenancy concerns. Such mechanisms need to be assessed in the context of security as well as in other properties such as performance, availability, manageability, and configurability. Effective and efficient data sharing across isolation mechanisms should not be overlooked.
Second, a side-channel attack is still a frequent vulnerability primarily due to multi-tenant paradigm. A side-channel attack means any attack is based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weakness in algorithm. The thing to note is that while a proper isolation can achieve security protection among tenants, existing countermeasures may not be applicable to a new side-channel attack; the appropriate countermeasure to address a particular side-channel attack may be too specific.

Third, due to the growth in cloud applications dealing with sensitive data, protection of sensitive data saved in a cloud (PaaS specifically) should be a major concern. Encryption in motion and at rest is one way of managing the problem. As data proliferate, so do regulations and policies attempting to protect the data, particularly personal data. There need to be explicit and flexible services that can deal with import/export controls, and different and changing policies and regulations. For example, EU has issued EU Data Protection Directive\(^6\). Organizations or governmental agencies

\(^6\)http://ec.europa.eu/justice/data-protection/index_en.htm
have to comply with such polices. However, storage capacity may reside in the cloud [61]. As the era of big data has inevitably arrived, we may expect to see more robust methods of data geolocation and protection to cope with increasingly more stringent requirements.
5.1 Introduction

Scientific WorkFlow Management Systems (SWFMS) have been widely adopted in scientific computing. The Kepler scientific workflow system (Kepler)\(^1\) is one example of that. It simplifies the effort required to create executable models of workflows by using GUI (graphical user interface) [81]. Kepler is data-flow oriented, and is used for a range of applications, such as bioinformatics (bioKepler), wildfire analysis (WIFIRE), microbial ecology research (CAMERA), biomedical, etc [82]. While Kepler can run and execute workflows on a single laptop, it is typically used to run workflow elements on distributed resources, such as high-performance computers, separate visualization resources, etc. Kepler also runs well in cloud environments into which more and more of the scientific computing

\(^1\)The Kepler Project: https://kepler-project.org/
5.2 RELATED WORK

is migrating [154, 166]. Migrating SWFMS from traditional Grid computing environments into cloud environments provides both scalability and elasticity with respect to resources, something that can be very cost-effective for large scale scientific workflows [167]. However, it opens up issues such as multi-tenancy, data protection, location and privacy, etc. Security must be an important factor when taking scientific applications into the cloud [58]. Unfortunately, Kepler does not have an explicit security engine yet. It basically relies on the security functions and service of the system in which it runs - for example, VLANs, VPNs, VM isolation, access control, etc.

Kepler embraces provenance as an important component that can help scientist ensure traceability and reproducibility of their scientific analyses and processes [42, 107]. The Kepler Provenance module stores provenance information in a database during workflow execution [8, 59]. Data provenance saves intermediate and end results that can be used to trace the data and its derivatives (meta-data). Process provenance, on the other hand, saves data and parameters during a workflow execution. Such information is used to validate processes [104, 109].

The goal of this study is to explore data-flow oriented elements needed for securing Kepler. We first define necessary security properties, then extend Kepler provenance model into a security model, and finally implement and assess a prototype.

The chapter is organized as follows. Section 5.2 presents related work. Section 5.3 provides problem statement, an analysis of Kepler and Kepler provenance module, and a discussion of the threat model. Section 5.4 presents the design of our secure Kepler (S-Kepler). Section 5.5 discusses our prototype implementation. Section 5.6 provides a discussion of the security.

5.2 Related Work

There is a wide variety of SWFMS for business and scientific domains [21, 39, 167]. Many execute in clouds. A list is in Table 5.1. Some of the SWFMS have features that could be used either explicitly or implicitly to implement run time security. For example, Taverna [159] provides authentication;
5.2. RELATED WORK

<table>
<thead>
<tr>
<th>SWFMS</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airavata</td>
<td><a href="https://airavata.apache.org">https://airavata.apache.org</a></td>
</tr>
<tr>
<td>Askalon</td>
<td><a href="http://www.askalon.org">http://www.askalon.org</a></td>
</tr>
<tr>
<td>Kepler</td>
<td><a href="https://kepler-project.org">https://kepler-project.org</a></td>
</tr>
<tr>
<td>Pegasus</td>
<td><a href="https://pegasus.isi.edu">https://pegasus.isi.edu</a></td>
</tr>
<tr>
<td>SCIRun</td>
<td><a href="http://www.sci.utah.edu/cibc-software/scirun.html">http://www.sci.utah.edu/cibc-software/scirun.html</a></td>
</tr>
<tr>
<td>Tavaxy</td>
<td><a href="http://www.tavaxy.org">http://www.tavaxy.org</a></td>
</tr>
<tr>
<td>Taverna</td>
<td><a href="http://www.taverna.org.uk">http://www.taverna.org.uk</a></td>
</tr>
<tr>
<td>Triana</td>
<td><a href="http://www.trianacode.org">http://www.trianacode.org</a></td>
</tr>
<tr>
<td>Trident</td>
<td><a href="https://www.microsoft.com/mscorp/tc/trident.mspx">https://www.microsoft.com/mscorp/tc/trident.mspx</a></td>
</tr>
</tbody>
</table>

Table 5.1 A list of Scientific Workflow Management Systems (SWFMS) in broader use

it can be used to access various secure service via both HTTPS and HTTP with WS-Security for securing Web services. Several SWFMS such as Airavata [101], Askalon [54], Kepler [9], Pegasus [43], Taverna [159], Triana [146], and Trident [20] have provenance management tools that monitor the workflow and intermediate results. Some SWFMS such as Pegasus and Trident also provide fault tolerance. Kepler also has that option [36, 108]. Tavaxy is running on Amazon Cloud Computing platform where each user has an independent Tavaxy system on independent machines, which may support secure multi-tenant environments [129].

Research and development activities related to provenance are divided into two classes. The first is about improving functionality of provenance module(s), such as collection, semantic analysis and dissemination of provenance information. Moreau et al. [107] address provenance in SWFMS and the Open Provenance Model\(^2\) that is designed to meet the requirements of provenance information. Provenance information could be used for many purposes, such as querying input/output associations, verifying results, and fault tolerance [36], but also for security purposes. Davidson and Freire [42] address several provenance models for SWFMS and discuss open problems such as information management infrastructure for the growing volume of raw data, efficient processing for analyzing and visualizing data and connecting database and workflow provenance. The other

\(^2\)http://openprovenance.org
activity is for securing provenance data. Cheney [33] proposes formalization of security properties for secure provenance. Abbadi and Lyle [2] propose the extended provenance (i.e., cloud provenance) to establish trustworthy cloud computing provenance.

While provenance information has so far been used with success for fault detection and tolerance, to the best of knowledge, it was not used to much to manage security vulnerabilities in SWFMS.

5.3 Kepler

5.3.1 Problem Statement

Kepler does not have any explicit functionalities to secure itself. It is relying on security features of the system where Kepler is executed. Unfortunately, as long as such systems are connected via a network, Kepler is open to the risk of damage caused by an adversary. What can help is adding a Kepler security analysis package (we call it K-SAP) that would alert Kepler workflow owner of any issues, especially when Kepler is executing partially or in full in a cloud.

Kepler has the provenance module that provides functionality for capturing and querying workflow execution history, specifically as it relates to data flows. It stores data and artifacts generated by application execution in a database. Analysis of provenance data may provide a clue of suspicious activities targeted at computing and communication resources. Then, we have the practical question: “Can the provenance module be used for securing Kepler workflows?” We believe that the answer is “Yes”, provided the provenance collection scope is somewhat broadened because:

1. All provenance information needed for adequate security analysis is not accommodated in the existing database.
2. The provenance information in the database is not sufficiently fine-grained.
5.3.2 The Analysis of Kepler

Kepler is a data-flow oriented engine. It has options to execute different process models. Modules that provide that functionality are called Directors. Information on data and data-flows (as opposed to control flows) is what is primarily available. Therefore, security analyses will be primarily data-flow oriented. Also, we focus ONLY on the security of the Kepler workflow and with respect to flows coming in and out of the workflow, not of the remote resources it may be using to run its computations. However, we do want to validate external flows and have assurances that external (remote) resources are not compromised.

5.3.2.1 Security Properties of Kepler

We started by analyzing Kepler “actors” (or processes) [80]. Actors are interconnected by data-flow pipes. As already mentioned, flows and transformations (including parallelism if any) are controlled by the workflow “director”. Kepler has a range of directors - for example SDF (Synchronous Dataflow; serial flows), PN (Process Network flows), and DDF (Dynamic Dataflow).

We have analyzed all actors and directors (a total of 345 components) that ship with Kepler’s standard library. As described in the Kepler Actor Reference [80], actors are categorized according to the functionality such as data input, data operation, data output, file system, etc. Each actor behaves as an independent function (see Figure 2.2) such as a typical method (or function) in programming languages; it consists of three parts: (1) inputs (e.g., input ports and parameters); (2) the method body; and (3) outputs (e.g., output ports). Inputs and outputs communicate to neighboring actors, but methods could be self-contained, or they could communicate to entities outside the actor. If it is assumed that an actor is inherently sound (and if it is not, that should be detected), then the primary security checking would occur in input, output and method-external flows. We focus on the parameters of each actor because such parameters are of the features to be executed. For example, the FileReader actor has two parameters (i.e., newline and fileOrURL); the actor reads the file
via fileOrURL. If the file should not be altered during workflow execution; data integrity should be ensured.

We recognize three primary data-flow oriented security properties. **Input validation** is to detect unauthorized input before it is processed by the application (or Kepler actor or workflow node). One of the most frequent security errors in software today is related to lack of input validation [106]. Actors in data input category accept various kinds of data, such as file name, path, URL, and executable or system commands, without performing input validation. For example, the ExternalExecution actor can be used to launch an external application from within a Kepler flow. The command parameter in the ExternalExecution actor should be verified. Untrusted input may allow attackers to execute unexpected, dangerous commands directly on the system [106].

**Remote access validation** is to protect Kepler from accessing malicious remote resources. When remote resources are necessary, Kepler utilizes remote access. Many actors in Kepler already support remote access without performing validation processing. For example, the WSWithComplexTypes actor invokes web services defined by WSDLs (Web Services Description Language). Accessing invalid WSDLs or broken URL links may lead to invalidated redirects and forwards attacks [117].

**Data integrity** is to maintain the accuracy and consistency of data during workflow execution. Kepler projects use various kinds of data. When used as read-only input data and input files, the data should be the same as when it was originally recorded. For example, the FileReader actor reads a local file or URL and outputs the contents of the file as a single string. The contents of the file should not be altered during execution. Data integrity is one of the most critical elements that Kepler needs to protect when operating in the cloud [139].

5.3.2.2 **Kepler Provenance module**

We analyzed the implementation of Kepler provenance module version 2.4 in terms of security because of our expectation that it may be applicable to secure Kepler or S-Kepler [59].

Provenance information is written in several formats (i.e., to a database, as a text file, and as an
XML file). For a database, the relational database schema represents three types of information [59]: (1) the contents or specification of workflows; (2) how these specifications change over time; and (3) events that occur during a workflow execution. Table 5.2 shows all tables and most important fields related to their use for verification with the provenance-2.4 database [59]. Several notable fields in Table 5.2 related to security information are:

- `parameter`: value contains URL, text input, file input, and command line
- `workflow_change`: time records time of a change of workflow
- `workflow_exec`: start_time and end_time records time of execution

Such information may be used in the assessment of the data-flow security of a workflow. Unfor-
5.3. KEPLER

Unfortunately, it is difficult to extract the precise information which we need for security verification given the current implementation of the provenance database. Modifications were needed. Several examples in Section 5.5 illustrate how the data is stored in the Kepler database.

![Diagram](image.png)

**Figure 5.1** Trust assumption in the Kepler threat model

### 5.3.3 Threat Model

We assume that (1) Kepler engine (application, its actors, its directors and its database), including provenance information, is secure, (2) the functionality of each component is secure and (3) the adversary may have access to remote resources that Kepler needs to access to execute a workflow.
Figure 5.2 The Kepler System Architecture with the Security Analysis Package (K-SAP) added.

Figure 5.1 illustrates the trust zone we assume - workflow engine and control plane. An adversary may launch both external and insider attacks. In external attacks, the adversary may access a vulnerable remote resource in a kind of a watering hole attack [142]. The adversary may also compromise some internal resources. In general, an adversary may seek to (1) alter files which Kepler should use, (2) send incorrect inputs to either distort and shift operational profile and thus affect one or more actors, or invoke malicious codes.

5.4 Secure Kepler (S-Kepler)

The solution we have implemented is to use a somewhat modified Kepler provenance module add Security Analysis Package (K-SAP) that analyzes provenance information in the security context.

Figure 5.2 shows the Kepler system architecture illustrated in [7] by Altintas et al. K-SAP is integrated with provenance framework. Figure 5.3 gives an abstract overview of the S-Kepler state
transitions that verify the security of Kepler workflows. We now define five steps to explain how K-SAP works:

- **Starting Kepler**: Kepler is started.

- **Adding & Configuring PR (Provenance Recorder) and actors**: Once PR is added and configured, the security verification table is set up in a database with other provenance tables.

- **Running a workflow**: When running actors on a workflow, K-SAP (1) assures security properties (i.e., input validation, remote access validation, and data integrity), (2) processes security validation based on security properties, and (3) saves the results in the database.
5.4. SECURE KEPLER (S-KEPLER)  

Table 5.3 A security verification table in S-Kepler provenance database

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int</td>
<td>A primary key</td>
</tr>
<tr>
<td>type</td>
<td>varchar</td>
<td>A value to identify an actor (e.g., actor name concatenating parameter's name)</td>
</tr>
<tr>
<td>value</td>
<td>varchar</td>
<td>A value to verify security (e.g., hash value (MD5) for file, input value, URL, etc.)</td>
</tr>
<tr>
<td>property</td>
<td>varchar</td>
<td>Security property (e.g., data for data integrity, input for input validation, and remote for remote access validation)</td>
</tr>
<tr>
<td>customize_name</td>
<td>varchar</td>
<td>An actor's name defined by a user (e.g., MyStringConstant)</td>
</tr>
<tr>
<td>actor_name</td>
<td>varchar</td>
<td>An actor's class name in a source code (e.g., ptolemy.actor.lib.StringConst)</td>
</tr>
<tr>
<td>md5</td>
<td>varchar</td>
<td>A hash value for a resource (e.g., input files or URLs)</td>
</tr>
<tr>
<td>valid_result</td>
<td>int</td>
<td>Current valid results being used in a workflow and previous verified results</td>
</tr>
</tbody>
</table>

- **Finishing a workflow**: Once processing on a workflow is finished, K-SAP (1) conducts the data integrity check and (2) sends the results of both data integrity checks and other security properties from the database to the display window. It is important to note that during testing phase of the workflow one could let the all security checks happen, and then have any violations reported. However, in production, workflow needs to stop and examine any anomalies in order to assure that the workflow does not continue in corrupted state.

- **Terminating Kepler execution**: Kepler terminates all processing of execution.

Two blue ovals of K-SAP in Figure 5.3 are processed differently according to the security property, which is explained in detail in Sections 5.4.2, 5.4.3, and 5.4.4.

### 5.4.1 Security Verification Table

Data related to security verification of a workflow is collected in the security verification table. To do this, we modified Kepler provenance module. The security verification table
is illustrated in Table 5.3. Specifically, the valid_result field contains two bits. The first bit represents the verification result for both input validation and remote access validation. The second bit is a validity bit that indicates whether or not such a value is being used in the current workflow; the security verification table stores all parameters with validation results whenever a change of values of the parameter in each actor occurs.

5.4.2 Input Validation

K-SAP needs a feature that detects and filters unauthorized input. For example, using a whitelist of acceptable inputs may be useful and efficient unless the input of WFMS is very diverse [86]; this will indirectly limit the scope of an attack [120]. Kepler is used by specific users such as scientists, quantitative analysts, and workflow engineers so the input may not be diverse in the same domain. With the whitelist, Kepler can perform input validation with less time complexity based on the input type. Defining very strong validation pattern based on regular expressions could be employed [115].

We have implemented our prototype with input validation using a whitelist. The steps for input validation are as follows:

- Step 1. A Kepler application executes.
- Step 2. Provenance module extracts the information from invoked actors.
- Step 4. K-SAP compares the input values (i.e., and input security properties) with the gating test (e.g., whitelist).
- Step 5. The result is sent to the security verification table in the database.
- Step 6. In production, any anomalies detected result in workflow pause, and evaluation of the impact, and a decision whether to continue or not (Figure 2.4).
5.4.3 Remote Access Validation

In addition to a firewall that detects invalid or outbound addresses and ports, there should be an application firewall. invalid inbound and outbound traffic. In addition, a link checking service may be used to find broken URLs. For example, Link Checker³ is useful in that respect.

We have implemented an internal firewall. Kepler application firewall limits access to valid URLs that Kepler workflows can access remotely. The steps for remote access validation are the same as for the input validation in Section 5.4.2.

5.4.4 Data Integrity

K-SAP monitors data integrity. For parts of the data that is supposed to be invariant, the routine should compare before-and-after Kepler execution. A hash value (e.g., MD5⁴) for such data can be sent to the security verification table in the provenance database.

Figure 5.4 shows the timeline for processing data integrity. We define the two time points (i.e., on-hash and post-hash) to check data integrity. “on-hash” is the time when data is being used in a workflow. “post-hash” is the time when the execution is terminated. The steps for checking data integrity are as follows:

• Step 1. A Kepler application is executed.

• Step 2. Provenance module extracts the information from invoked actors.

• Step 3. K-SAP determines the security property of the information.

• Step 4. K-SAP generates the MD5 value of data used by the actor.

• Step 5. The MD5 value is sent to the security verification table in the database.

³http://validator.w3.org/checklink
⁴The MD5 Message-Digest Algorithm; http://tools.ietf.org/html/rfc1321
Figure 5.4 The timeline for processing data integrity

- Step 6. The Kepler workflow is finished.
- Step 7. K-SAP extracts the MD5 value (i.e., on-hash) and file location information (e.g., file directory) from the database.
- Step 8. K-SAP compares the MD5 value (i.e., on-hash) from the database with the MD5 value (i.e., post-hash) generated from the file pointed by file location information in a workflow.
- Step 9. The result is saved in a buffer to display results.

5.5 Prototype

5.5.1 Before and After implementation of K-SAP

Our pilot is based on Kepler 2.4 [81]. We illustrate using several sample workflows. Each workflow has data workflows related to the three security properties we discussed earlier in Section 5.3.2.1. We compare before and after K-SAP situation to show how the state of each security property can be recognized by K-SAP.
5.5. PROTOTYPE

CHAPTER 5. A CASE STUDY: KEPLER

(a) Before
(b) After

Figure 5.5 Input validation: Executing an external application by a command from an actor
5.5. PROTOTYPE

CHAPTER 5. A CASE STUDY: KEPLER

(a) Before
Figure 5.6 Remote access validation: Accessing a remote database using a web service application
5.5. Prototype

(a) Before
(b) After

Figure 5.7 Data integrity: Using local data
5.5. PROTOTYPE

CHAPTER 5. A CASE STUDY: KEPLER

(a) Visual interface
Figure 5.8 The execution result of a secure workflow with K-SAP
(a) Visual interface
Figure 5.9 The execution result of an insecure workflow with K-SAP
Before we continue though, we would like to draw attention of the reader to Figure 5.5a to explain Kepler GUI and some of its simpler actors. Looking at the top of the figure, we see that the process controller (or Director) used is SDF, so flow occurs serially (they execute one actor at the time). Provenance Recorder has been attached. The workflow consists of an input string that in this example is hardcoded but could also have come from another actor or an external source, an actor called External Execution that will take the input and then execute that input (in this case a command that invokes local HelloWorld program) on the machine local to the Kepler engine, and an actor that will then take the returned value from that command (e.g., stdout) and show it on the screen. Also shown (in the middle) is a more detailed description of the “CommandLine” actor, the library it belongs to (in this case constant) and its location information. Finally, the bottom part of the figure shows provenance database entries. 23 in id field is the content of the Constant actor and 46 in id field is what enters the External Execution actor.

Input validation: Figure 5.5a and 5.5b show that a workflow executes an external application by a command from an actor; it is vulnerable to injection of flawed data which is the top category among SaaS vulnerabilities [86]. Figure 5.5a shows the present implementation of provenance module (i.e., “Before” K-SAP implementation). Again, it includes three areas: (1) the workflow (top), (2) the configuration of a Constant actor (middle), and (3) the provenance information stored in a MySQL database (bottom). This workflow uses Kepler’s ExternalExecution actor to invoke the HelloWorld application that ships with Kepler [81]. If an attacker can inject a non-legitimate command into the Constant actor, attacker can execute a malicious external application in the local computer by typing unexpected commands [106]. Note that the input value for the CommandLine actor in Figure 5.5 has a strict rule in order to execute Java applications. For example, the “-cp ./” part of command tells Java to include the current directory in the Java classpath [81]. In addition, there is one space between “./” and “HelloWorld” in this input value; Kepler users may be puzzled unless they recognize such a rule. Input validation processing may help Kepler users in this case. To conduct input validation, we may find the input value (i.e.,
java -cp ./ HelloWorld Kepler_User) in the parameter table in the provenance database. However, there are two different entries of this value (i.e., ptolemy.data.expr.Parameter and ptolemy.actor.parameters.PortParameter) in the red dotted box of Figure 5.5a because they are created by two actors (i.e., Constant and ExternalExecution). This may cause confusion. On the other hand, if the two values are different, it should be cause for concern. Figure 5.5b shows the modified provenance module (with security table). K-SAP validates input value (i.e., java -cp ./ HelloWorld Kepler_User) perhaps using the whitelist. K-SAP stores the result in the security verification table and then displays the result after finishing the workflow execution. Red arrows point to the result of input validation and input value stored in the security verification table in the provenance database.

Remote access validation: Figure 5.6 illustrates how a workflow invokes and external a web service application; it represents the vulnerability of remote URL access. Figure 5.6a shows the “out-of-box” implementation of provenance module (i.e., “Before” K-SAP implementation). It includes three windows: (1) the workflow (top), (2) the configuration of a WSWithComplexTypes actor (middle), and (3) the provenance information stored in a MySQL database (bottom). This workflow uses the WSWithComplexTypes actor to access a remote genomics data service to retrieve gene ID from its gene name via the URL (i.e., http://npd.hgu.mrc.ac.uk/soap/npd.wsdl) [81]. To check the vulnerabilities of the URL by user-controlled input that specifies a link to an external site, we have to extract the URL from a database. The type of the URL value in a parameter table is ptolemy.actor.parameters.PortParameter in the red dotted box, which seems to be a unique type. However, it is still difficult to extract the URL from the database because the type of URL value is created from a code level property; Kepler users may not recognize a URL value with a type. The configuration property of user interface (i.e., WSWithComplexTypes.parameters.wsdl) may be better for that, similar to the previous example of Figure 5.5b. Kepler does not have any security module to verify URLs but fully depends on system security such as a firewall for network security. If a security analysis platform enables to scan web applications such as IBM Security AppScan...
5.5. PROTOTYPE

Standard [73, 106], it may mitigate malicious input vulnerabilities of remote URL access. While relying on external security services and host system features may work, it is probably much better to have some basic level of security built into a complex application, or a control application such as Kepler. That not only gives the workflow developers more security awareness, but also acts as a safety net in the case external security fails. Figure 5.6b shows remote access validation processed by K-SAP. Once K-SAP recognizes the URL, it scans Kepler internal firewall which has the valid list such as IP addresses and URLs; it may allow Kepler workflows to access a valid URL.

**Data integrity:** Figure 5.7 shows a simple workflow that uses a local input file; it illustrates the vulnerability of data integrity. Figure 5.7a shows the File Reader actor reading a data file and sending the data to the Display actor. Kepler provenance module stores provenance information in a database. However, in its current (standard) incarnation provenance database of Figure 5.7a does not collect information about the content of the local files. The workflow itself is part of the record, so from the perspective of understanding the workflow, all information is there. Unfortunately, there is no way of checking data integrity. Kepler provenance module should have a feature that takes data information (e.g., the hash value of such data file) for data integrity and then stores it to a provenance database so that K-SAP can compare hash values before and after (i.e., on-hash and post-hash in Section 5.4.4) workflow execution. Figure 5.7b shows what K-SAP does. When a provenance module recognizes data file as an input for the File Reader actor, K-SAP stores the hash value of data file in the security verification table. As soon as a workflow is executed completely, K-SAP compares on-hash and post-hash and then displays the result of data integrity. Red arrows point to the result of data integrity and the hash value stored in the security verification table in the provenance database.

5.5.2 A Prototype of Kepler Security Analysis Package (K-SAP)

Workflow shown in Figure 5.8 is being checked for three security properties that K-SAP is able to detect; it has been modified from Figure 5.6 by adding the File Reader actor. Figure 5.8 shows the
execution results of a secure workflow with K-SAP. Figure 5.8a displays (1) several output windows from a workflow and (2) the Secure Kepler Testing window in the bottom-left corner. The Secure Kepler Testing window consists of two parts. The first part is to display the result of security validation; the first bar displays the overall result for the workflow and the other bars display the result of each security property. The second part is to display error logs. Each error log has customize name, actor name, and value that are stored in a database. Figure 5.8b shows the details in the database.

The two figures show the contrast between a valid input (Figure 5.8) and an invalid input (Figure 5.9). Figure 5.8 represents that S-Kepler executes a workflow with K-SAP securely. On the contrary, Figure 5.9 represents that S-Kepler executes a workflow with K-SAP where the given workflow is “insecure”. Instead of a valid input (i.e., ATRX) in Gene Name actor, the invalid input (i.e., ATRX-wrong) has been added. The workflow has been executed. Once K-SAP detects the vulnerability, it displays “Not Secured” with red box for overall workflow and input validation and error logs extracted from the security verification table in database in Figure 5.9a. There are two dotted arrows coming out of the Gene Name actor in Figure 5.9b. The left arrow points the output with error messages produced by a given workflow due to an invalid input. The right arrow points the database where the invalid input is saved.

The thing to remember in this context is that the Kepler engine does not have to run in the cloud, it can run on a secure laptop of a scientist. Therefore the engine and the control plane can be inherently secure provided that data-flow inputs and outputs are checked for unusual behaviors. On the other hand, the “heavy lifting” - simulations, Internet searches, etc. are conducted in the cloud (to reduce costs). What moves between the cloud and the control engine need only be low-volume parameters and meta-data that are checked for security via K-SAP.
5.6 K-SAP as a Testing Tool

We need to remember that the prototype, as we are using it in our examples, is intended as a testing tool so we run through all inputs and verifications and report results at the end. In production, analysis would be done in real time as would anomaly impact adjudication.

Figure 5.10 A Kepler GUI showing a workflow with $f_i$ labels

Figure 5.10 is the screen shot of a Kepler GUI showing basic Kepler sample workflow we showed in 5.8 where we added labels for functions (e.g., $f_i$). It has 7 actors (or components) and 6 data flows,
one of them is an external communication channel (to a cloud service). In this example, Kepler workflow engine is operating under a single-thread synchronous model (SDF Director), and both Provenance Recorder and Security Analysis Package are active. Input actor \( f_1 \) emits string ATRX. Second actor, \( f_2 \), sends it to Nuclear Protein Database API in UK via its SOAP interface. It receives back ATRX meta-data. That is processed by \( f_3 \) and the original input is displayed by \( f_5 \). \( f_4 \) receives extracted geneID and displays it. It passes the information to File reader actor \( f_6 \) that then goes to local gene data bank to get more information. That information is also displayed.

Figure 5.11 Workflow validation

Figure 5.11 illustrates security validation analysis of dataflows shown in Figure 5.8. Each component may or may not have security properties to validate. The workflow includes three security properties—input validation both in \( f_1 \) (String Constant actor) and \( f_5 \) (File Reader actor), remote access validation in \( f_2 \) (WSWithComplexTypes actor), and data integrity in \( f_3 \) (File Reader actor). Each component may access either local or remote (cloud) resources. Little circles on \( f_1 \),
where Kepler security analysis package (K-SAP) explicitly validates security properties. Label “T” indicates implied validation through security properties which stay protected in internal operations as long as the workflow accepts valid data. Validation of security of the workflow is different from validation of its correctness. In the case of correctness, we would care whether ATRX should really be ATRX or something else. In the case of security we primarily worry about the format and range (e.g., four capital letters) of the first input so no damage would result. Figure 5.11 shows this concern as the first \( f_1 \) input validation point (via a regular expression in this case). Since we know that we wish to get gene meta-data from a web-service that we trust, we send validated request to that URL after we have validated URL authenticity and soundness (via certificate, \( f_2 \) validation point). We then trust the external output we receive. After that we do not explicitly validate for security since we trust the workflow until we need to read from a file \( (f_6) \). At that point we check integrity of the data (file) using MD5. If it is validated, we continue the workflow.

We represent the workflow in Figure 5.11 with Equation 2.3 (Section 2.2) (i.e., 5-tuple \( WM =< F, O, C, SP, SC > \)) as follows:

\[
\begin{align*}
F & = \{ f_1, f_2, f_3, f_4, f_5, f_6, f_7 \} \text{ (i.e., actors in the workflow)} \\
O & = \{ o_1, o_2, o_3 \} \text{ (i.e., User input, Remote Genomics DB, Local data)} \\
C & = \\
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\end{align*}
\]
5.6. K-SAP AS A TESTING TOOL

\[ SP (\text{Security Property}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \]

where each column indicates security property (i.e., input, remote, data integrity) at each function that should be verified (i.e., \( f_1 \) to \( f_7 \)).

\[ SC (\text{Security Class}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \]

where (1) SC starts in initial state \( SC_{init} \in SC \) (corresponding to input validation, remote access validation, and data integrity validation states - 1 for OK; 0 for implied or not-applicable) and (2) each column indicates security class at each function (i.e., \( f_1 \) to \( f_7 \)).

As the workflow executes we perform run-time validation at critical points and we keep a score of successful validations and of implied validations. This trace is stored. The validation result can be obtained simply by the matrix result of \( SP - SC \). If the matrix is \textit{null} matrix at the last state (i.e., terminating Kepler execution in Figure 5.3), the workflow execution is secure.
We show practical approach by keeping Table 5.4 in K-SAP where each component has a 3 bit state (corresponding to input validation, remote access validation, and data integrity validation states - 1 for OK; 0 for implied or not-applicable). The last column indicates that expected secure state vector. An XOR between actual state vector and expected vector yields a go/no go bit (0 for OK, go; 1 for possible violation). If the check are all 0s (zeros), the workflow is secure (in the sense of the defined properties). Otherwise, we have detected an anomaly that will need to be assessed. For example, in Table 5.4, $3 \text{ bit } \oplus \text{ Secure } \rightarrow \text{ All 0s}$; this indicates that a workflow in Figure 5.8 has its security properties preserved to the extent the out checks have validated that.
A practical question is the overhead that is encountered by security a workflow platform such as Kepler. We assess run-time overhead of K-SAP implementation discussed in Chapter 5. Our Kepler Security Analysis Package (K-SAP) leverages Kepler’s Provenance Recorder (PR). K-SAP secures Kepler data flows from external input-based attacks, from access to unauthorized external sites, and from data integrity issues. Therefore, it is not a surprise that cost of real-time security is a certain amount of run-time overhead. About half of the overhead appears to come from the use of the Kepler Provenance Recorder and the other half from security function added by K-SAP.
6.1 Introduction

Kepler embraces provenance to keep track of data and processes [7]. We leverage its Provenance Recorder (PR) to capture workflow execution history and leverage this to implement pro-active data-flow security. We have implemented a prototype of what we call Secure Kepler in Chapter 5 where we modified Kepler provenance module and added to it our Kepler Security Analysis Package (K-SAP) that analyzes provenance information in the security context [85]. Figure 6.1 illustrates Kepler workflow management systems with K-SAP.

K-SAP solution was discussed in detail in Chapter 5 [85]. In this chapter we discuss the overhead that solution may carry. The chapter is organized as follows. Section 6.2 provides background
6.2 Background

6.2.1 Provenance

Several SWFMS have provenance management tools [85]. Typically provenance systems will consist of a data collection modules, and data (or meta data) storage modules. Latter are often databases. Pegasus uses a lightweight job monitoring module called pegasus-kickstart [44, 153]. The Taverna Provenance suite also records service invocations, intermediate and final workflow result in the Open Provenance Model format [159]. Triana provenance records provenance related information [99]. However, they do not appear to be report measured run-time overhead of provenance capture for Taverna workflow engines. On the other hand, Trident workbench analyzes performance of its provenance. Trident uses its native provenance model as well as Open Provenance Model (OPM). They mention that run-time overhead may be caused by different databases used as part of OPM and the native provenance collection operating at runtime, and the storage overhead may occur due to duplicate copies of provenance metadata. The run-time overhead Trident observes affects scalability [137].

Performance of each added security and provenance related module (e.g., hash function and database access process) that is used for K-SAP needs to be tuned. Performance of hash functions such as MD5, SHA-1, BLAKE2, etc in discussed in [16]. MD5 is widely used [128]. Kepler uses MD5 algorithm to test the integrity of files [83]. Similarly OpenStack cloud storage system, revision control system Perforce, and object storage system in AOL all rely on MD5 for data integrity [16]. Many are working on improving database performance in both logical (i.e., SQL statement and table structure) and physical (i.e., index design) spaces [113]. However, performance overhead can surprise when all modules are integrated into a system.
6.2. BACKGROUND

6.2.2 Secure Kepler

On its own, out-of-the-box Kepler does not have any security modules and it relies on security features of the system where Kepler is executing. We have implemented a prototype of secure Kepler with Security Analysis Package (K-SAP). K-SAP is intended to secure input/output flows at the level of the control plane [85]. K-SAP uses Kepler's provenance module to collect data.

There are three data-flow oriented security property of interest in our model. **Input validation** is used to detect invalid or unauthorized inputs before they are processed by the application (or actor). Some of the most frequent security errors are related to lack of input validation [106]. **Remote access validation** protects Kepler from accessing malicious remote resources. Kepler applications may access external resources and some of those may be compromised. For example, accessing invalid URLs or broken links may lead to invalidated redirects and may invite external attacks [117]. **Data integrity** tests ensure that internal data flows are protected from accessing data stores that may have undergone unintentional or unauthorized changes (e.g., files). When input data and input files are used in read-only modes, they should be the same as when originally recorded.

In this model, we assume that it is possible to secure the Kepler engine itself (application, its actors, its directors and its database) as well as its provenance module and the information the module collects. However, we allow that an adversary may have access to remote resources that Kepler may use to execute a workflow. In other words, we assume that the top two layers shown in Figure 1 are running in a secure space, and that corruption or attacks if any emanate from the lowest level - external resources. We therefore protect workflow points where data/information flows into a workflow either from some external source, or from an internal data store (e.g., file). We trust the actors that receive inputs only from internal data flows. In complex real-life workflows there may be thousands of actors [95] but only relatively few may be accessing external information and data stores. In that case, while per I/O active actor security analysis overhead may be substantial, the average over the whole workflow may be roughly proportional to the cost of provenance data...
K-SAP “knows” what security properties, if any, should be verified for an actor based on security verification/validation table added to the provenance store. For example, K-SAP could use a whitelist to check for acceptable inputs, or to check whether the actor is allowed to access certain external resources. K-SAP compares the input value (i.e., input security property) with the whitelist of uses some other gating algorithm. For remote access validation, K-SAP works as an internal firewall which allows valid URLs and IP addresses that Kepler workflows are allowed to access remotely, and blocks other. For data integrity, K-SAP conducts a data integrity check before and after Kepler access to stored data. To reduce the overhead of integrity checks, we use a hash value (e.g., MD5). K-SAP stores the hash value in the provenance security table. For example, the Message Digest Test actor uses MD5 algorithm to compare a file to its previous version.

6.3 Run-time Overhead

So how much does it cost to secure Kepler inputs and data? Provenance Recorder (PR) was designed to keep track of the provenance of data and processes [7]. Performance overhead for PR may vary depending on the amount of provenance data created by a workflow. When a workflow uses PR, a provenance log is automatically created and added to provenance database during a workflow run. Figure 6.2 illustrates how K-SAP works with the operation of checking security properties. We define five steps to explain potential overhead: (1) Staring Kepler; (2) Configuring PR and a workflow K-SAP probes; (3) Executing a workflow; (4) Finishing a workflow; and (5) Terminating a workflow. K-SAP is set up with PR in Step 2 (e.g., setting up a security verification table in PR database). K-SAP is executed in Step 3 and 4 while PR is executed in Step 3. Once PR sends the information from invoked actors to K-SAP, K-SAP conducts its function based on an actor’s information. Input validation and Remote access validation are completed in Step 3 and Data integrity is completed in Step 4 after comparing on-hash and post-hash values. For example, I actor in Figure 6.2 is an actor.
for Input validation. K-SAP takes an input value, conducts input validation and then sends the validation result to the security verification table. For input validation, K-SAP compares an input value with values in the whitelist which may lead to a time complexity of $O(n^2)$, where $n$ is the size of the whitelist. Another example is D actor in Figure 6.2. K-SAP generates on-hash value and sends on-hash value and metadata information (e.g., file directory) to the security verification table. On finishing step 3 (Executing a workflow), K-SAP queries metadata information to the security verification table and generates post-hash value. K-SAP compares on-hash from the database with post-hash where a time complexity may be $O(n^2)$; this overhead can be high. With reference to Figure 6.2 the overhead ($O$) of K-SAP can be defined as:

$$O = \sum_{i=1}^{j} I_i + \sum_{i=1}^{k} R_i + \sum_{i=1}^{l} D_i + \delta$$ (6.1)
where (1) $I$, $R$, and $D$ is the overhead that occurs when we check on the security properties for input validation, remote access validation, and data integrity, respectively; (2) $j$, $k$, and $l$ are the number of actors that require checking of their security properties instead of assuming them. Typically those would be actors that communicate with exterior or with internal or external data stores; and (3) $\delta$ is the database access time.

The overhead of input and remote access validation may be smaller than that of checking data integrity because generating hash values may be a costly operation. In addition, K-SAP needs to generate two hash values (i.e., on-hash and post-hash) to compare before and after values. Of interest is, for example, (a) how much overhead can one expect when K-SAP is used, and (b) how is this overhead distributed, i.e., how much is the cost of a typical integrity check, a typical input validation, and a typical remote access check.

### 6.4 An Evaluation

For explanation purposes we use a very simple workflow illustrated in Figure 6.3. The workflow has one or more FileReader actors, and a Display actor. The changing element is the amount of data read. We use FileReader actor for our micro-benchmark because FileReader has two security properties to be verified - input validation and data integrity. First we check that the file has not been corrupted, then we check the inputs to make sure the data read from the file does not cause problems for the receiving applications, e.g., unexpected escape characters. If the file is external to the control plane environment, its integrity would need to be checked every time (even if it is intended to be a read-only file). The same is true of its input impact on the workflow. Kepler application provides the execution time (i.e., execution finished) shown in the bottom of the window in Figure 6.3. We compute and average execution time by running each workflow a number of times.
6.4. AN EVALUATION

CHAPTER 6. ASSESSING OVERHEAD OF SECURE KEPLER

(a) Different file sizes

92
(b) A workflow with five FileReader actors

Figure 6.3 A workflow to measure run-time overhead of K-SAP
6.4.1 Experimental Platform

Experiments were conducted in a single desktop computer. Processor is Intel Core Quad CPU 2.40Ghz and RAM is 8GB. We used Kepler 2.4 running on Windows 7 Enterprise (64-bit Operating system) for evaluating the performance overhead of K-SAP. Figure 6.3 is the screenshots of a workflow in order to measure performance overhead. Figure 6.3a has a FileReader actor which are assigned different file sizes from 2MB to 10MB. Figure 6.3b has five FileReader actors, where the number of FileReader actors has been increased from 1 (one) to 5 (five). The files are generated by a simple program of Dummy File Creator\(^1\).

6.4.2 Results

6.4.2.1 K-SAP Overhead

Figure 6.4 shows average workflow run-time in milliseconds (vertical axis). We measure three cases: (1) K-SAP, (2) NO K-SAP - without K-SAP (i.e., only with PR), and (3) NO Provenance - without PR. X-axis in Figure 6.4a indicates file sizes which the FileReader actor should read and X-axis in Figure 6.4b indicates the number of FileReader actors in a workflow, each of which reads a file 2MB in size. For example, “3x2MB” in Figure 6.4b denotes that a workflow has three FileReader actors, each of which reads a file with 2MB in size. Y-axis is the execution time (Time unit: millisecond).

We note that PR includes PR code execution time and database access time which we call “baseline operating cost”. Looking at Figure 6.4, we observe the following:

- Running K-SAP creates overhead which in this case is about as much as just running PR. In this case running both PR and K-SAP just roughly doubles execution time of the workflow.

- When comparing Figure 6.4a and 6.4b, we also notice that reading the same amount of data from one large file may be more expensive than reading that total amount from smaller files.

\(^1\)http://www.mynikko.com/dummy/
We have not explored this further, but we speculate that this may be related to I/O buffer sizes. For example, reading 10MB in 6.4a is costlier than reading 5x2MB in Figure 6.4b.

### 6.4.2.2 What is the cost of each security property?

In order to better understand K-SAP overhead, we looked closer at two security properties being checked. We conducted the testing with three cases: (1) Input validation (INPUT); (2) Data integrity and Input validation (DATA+INPUT); and (3) only K-SAP (Only K-SAP). Figure 6.5 shows the measurements for the same workflows shown in Figure 6.4 where two more lines (i.e., NO K-SAP and NO Provenance) are added we used in Figure 6.4.

In Figure 6.5, we observe that three cases (i.e., DATA+INPUT, INPUT, and Only K-SAP) are almost the same. Besides visual inspection of the data, we did a quick statistical check to assess how “close” the curves were. We found that core codes for input validation and data integrity checks have very low overhead compared to I/O related overhead. There is almost no difference among DATA+INPUT checks, INPUT checks and Only K-SAP. This is both good and bad news. Integrity check appear to be cheap, but input validation can be a necessary but very costly operation. It is necessary, given that malformed inputs are often primary sources of security failures [106], but input check can perhaps be avoided if a file has been checked off-line for input violations and at run-time is read-only and its integrity is intact. Of course, I/O is always more expensive than in-memory operations so it is not surprising that file reading incurs additional overhead. It is somewhat worrying that with PR turned on, Kepler run is perhaps twice as expensive (in this example) as it is without PR, and that if input validation is done rigorously that doubles that overhead. We will be examining more elaborate workflows and under different conditions to better understand this experimental result.

Figure 6.6 compares K-SAP with NO K-SAP (Provenance Only). We calculate performance overhead based on “No Provenance” as follows:

- \[ \text{Overhead}_{K-SAP} = \text{ExecutionTime}_{K-SAP} - \text{ExecutionTime}_{\text{NoProvenance}} \]
Figure 6.4 Measuring execution time during a workflow run
Figure 6.5 Run-time overhead: Input validation and Data Integrity
6.5 Summary

We have examined the level of run-time overhead incurred by our implementation of a security model for Kepler. There are two sources of direct overhead - one is use of the Kepler provenance recorder, the other one is run-time processes that K-SAP needs. These two costs are about the same. Interestingly, security checks themselves can be a relatively low cost activity. We continue to work on better understanding of the observed run-time overhead, and on finding a less "expensive" solution.

\[ \text{Overhead}_{\text{NoK-SAP}} = \text{Execution Time}_{\text{NoK-SAP}} - \text{Execution Time}_{\text{NoProvenance}} \]

where the \( \text{Execution Time} \) includes "Executing a workflow" and "Finishing a workflow" in Figure 6.2.

We observe that opening the secure Kepler testing window to show security results incurs overhead. Also, we observe that performance overhead is not confined to operation of checking security properties, but is part of K-SAP operation - which perhaps is not surprising since K-SAP draws data from PR even when not acting on any security properties. We are looking into possibly ameliorating the problem.

6.5 Summary

We have examined the level of run-time overhead incurred by our implementation of a security model for Kepler. There are two sources of direct overhead - one is use of the Kepler provenance recorder, the other one is run-time processes that K-SAP needs. These two costs are about the same. Interestingly, security checks themselves can be a relatively low cost activity. We continue to work on better understanding of the observed run-time overhead, and on finding a less "expensive" solution.
Figure 6.6 Run-time overhead: K-SAP vs. NO K-SAP
CHAPTER

7

SUMMARY AND FUTURE WORK

7.1 Summary

The goal of the research reported in this dissertation was to explore how to secure data-flows that are part of an application network (workflow) that may span a number of distributed resources, including clouds.

As part of this we (a) performed a comprehensive evaluation of security threats that may occur in modern distributed environments - specifically clouds, (b) we have developed a model for protecting data-flows of application networks, and (c) we have developed a working prototype of the solution using an open-source workflow management framework called Kepler.

Our model has three principal components: a workflow control plane and orchestration engine
that operates in a secured environment, but may (and often does) reach out to resources running
in a distributed fashion with special emphasis on clouds. We assume that flows within a secure
environment remain secure unless there is an invasion through externally facing channels and data
repositories. Therefore, to start with need to make sure when a secure workflow goes to external
services and resource, addresses to which it goes have been vetted and resource running at that end
provide some assurances about their security. Furthermore, integrity of data in repositories needs
to be ascertained. Externally facing channel flows need to be secured by making sure inputs are not
going to "break" the box. We advocate use of an extended operational profile model that ensures
that unusual or unexpected inputs (and outputs) do not occur. Implementations of that model
range from white and black lists, to a range of firewalls, to pattern matching, to implicit equations, to
pro-active diversity based anomaly detectors, etc. Some of these gating test may be quite complex.

In enhancing Kepler provenance collection capabilities for cybersecurity assessment, tracking,
and anomaly detection we have developed Kepler Security Analysis Package (K-SAP).

- Chapter 2 discusses assumptions and the model that we use to secure data-flows of an appli-
cation network.
- Chapters 3 and 4 discuss some of the current threats such networks face.
- Chapter 5 presents the prototype we have developed using the Kepler framework as the
  base-line.
- Chapter 6 briefly discusses run-time overhead that may be part of securing Kepler base
  workflows.

7.2 Future Work

We recognize that systems in cloud environments may require more complex frameworks than
Kepler can provide. We also recognize that this work has just scratched the surface of workflow
security assurance, and that our results, although very encouraging still need to be assessed in real production environments. However, the fundamental components of an individual software or system may not be different; a software application must have three parts: (1) input, (2) body, and (3) output. The system may reach the final goal securely as long as workflow input and output are secure under the assumption that each component is secure.

Future work includes extending our K-SAP into generic SAP (Security Analytics Package) for other scientific workflow managements or other systems in cloud environments. K-SAP has been implemented by modifying a provenance module integrated in Kepler. Likewise, in order to build a generic SAP, provenance-aware system may be utilized where a system (e.g., a monitoring system) can gather and report metadata that describe the history of each object processed [22]. A module to deal with security properties should be added based on the analysis of the system infrastructure (e.g., cloud environments). Building a generic SAP would be an exciting step forward for cloud security communities. Of course, assessment of the current model, as well as of future models and prototypes in production environments will also be part of future work.


[40] CVE. *heartbleed OpenSSL bug CVE-2014-0160*  


Publication notes

Parts of the work in this dissertation have appeared in the following publications:


Notations

\( \tilde{x}_i \): an input vector at function \( i \) (i.e., \( f_i \))

\( \tilde{y}_i \): an output vector at function \( i \) (i.e., \( f_i \))
LIST OF ACRONYMS

BYOD Bring Your Own Device
BYOI Bring Your Own Identity
COMM Communication
COMP Computation
HPC High Performance Computing
IaaS Infrastructure as a Service
Kepler Kepler Scientific Workflow Management Systems
K-SAP Kepler Security Analysis Package
NIST National Institute of Standards and Technology
OP Operational Profile
PaaS Platform as a Service
PR Provenance Recorder
S-Kepler Secure Kepler
SaaS Software as a Service
SAP Security Analysis Package
SWFMS Scientific Workflow Management Systems
VCL Virtual Computing Lab