The Pennsylvania State University
The Graduate School
College of Information Sciences and Technology

TOWARDS SECURE SOLUTIONS FOR CLOUD APPLICATIONS

A Dissertation in
Information Sciences and Technology
by
Cong Liao

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2018

The dissertation of Cong Liao was reviewed and approved by the following:

Anna Squicciarini
Associate Professor of Information Sciences and Technology
Dissertation Advisor, Chair of Committee

Peng Liu
Professor of Information Sciences and Technology

Sencun Zhu
Associate Professor of Computer Science and Engineering & Information Sciences and Technology

David Miller
Professor of Electrical Engineering

Mary Beth Rosson
Professor of Information Sciences and Technology
Director of Graduate Programs

Signatures are on file in the Graduate School.

Abstract

With the booming of Internet enabled services and technologies, data is being generated at a drastic velocity and volume. In light of such explosion of services and data, end users and industry players have begun to explore the potential of what big data could bring about if properly utilized. The rapid development of cloud computing further contributes to the advancement of big data analytics in the cloud. As a result, more and more data are being moved to the cloud with the proliferation of various cloud services and open source software that provide solutions for processing, storage and management of big data. In particular, data analytics based on machine learning prevails by harnessing the power of machine learning models to analyze massive data and succeeds in various challenging tasks including security critical applications.

However, the migration of data to the cloud leads to data security and privacy concerns as users generally do not have the absolute control regarding how their data are managed, which might not be beneficial to building the trust between cloud users and service providers. In addition, the application of machine learning models to sensitive data and services has led to new attack venues for adversaries to exploit, requiring further investigation to better understand vulnerabilities and improve the robustness of the learning models.

In this dissertation, we aim to mitigate these concerns, and to facilitate the trust between cloud users and service providers by enabling users with certain control in the process of data usage. Specifically, we look into Hadoop open software and present innovative mechanisms to extend user's control over how data flow in the course of storage in Hadoop distributed file system (HDFS) and processing with MapReduce. Further, we investigate vulnerabilities of machine learning models exposed in adversarial environment, and study two types of attacks against machine learning models in particular. Our main contributions in these three sub topics are as follows.

First, MapReduce enables parallel and distributed data processing in a cluster, but it is subject to threats posed by malicious workers that could tamper with the

data and computation. We investigate the computational data flow of MapReduce to detect anomalies in such process. Accordingly, we develop a computational provenance system that captures provenance data related to MapReduce computation within the MapReduce framework in Hadoop. In particular, we identify a set of invariants against aggregated provenance information, which are later analyzed to uncover anomalies indicating possible tampering of data and computation.

Second, cloud storage has gained increasing popularity in recent years. Despite its benefits, the lack of knowledge and control of the physical locations of data could raise legal and regulatory issues, especially for certain sensitive data that are governed by laws to remain within certain geographic boundaries. We study the data flow in Hadoop's underlying storage system, and address data placement control problem by supporting policy-driven location-aware storage in HDFS when files are uploaded and managed for load balancing purpose. User policy is consistently enforced and data movement is also monitored to detect data placement violation.

Third, machine learning models have been found to be vulnerable to well crafted sample as input causing its misclassification, e.g., evasion attack, or malicious samples contaminating the training process that leads to degradation of the modelâĂŹs efficacy, e.g., poisoning attack. Based on these known attacks, we explore two new attacks, named model manipulation and backdoor injection attacks, with the goal of causing the model to misclassify certain malicious samples and still perform well on normal ones. Specifically, the former attack chooses to manipulate the model parameters instead of the input sample. The latter attack instead tends to inject a backdoor, that can be exploited later by well crafted input, into the model by poisoning its training process with malicious samples.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my deepest gratitude and appreciation to my advisor Dr. Anna Squicciarini, for her tremendous support, patient guidance and valuable advice throughout my time studying at Penn State University as her student. I am extremely lucky to have her as my mentor both academically and professionally along the journey. Without her persistent help, this dissertation would not have been possible.

I would also like to thank Dr. Peng Liu, Dr. Sencun Zhu and Dr. David Miller for serving as my dissertation committee members. I truly appreciate their time and feedback on improving the quality of my research and dissertation. In particular, I would acknowledge Dr. Sencun Zhu who helped with my research during my advisor's sabbatical.

In addition, I would like to thank my friends and colleagues in the College of Information Sciences and Technology, who provided not only a much needed form of escape from my Ph.D. studies but also insights and inspirations that contribute to my research.

Finally, I am most grateful to my parents. My father and mother have provided me with unselfish love and continuous encouragement as I experience the ups and downs in research and life. Thanks to them, I am taught to have faith and perseverance that lead to successful completion of my Ph.D. studies at Penn State University.

This dissertation is supported in part by National Science Foundation under award No.1250319 and No.1421776

Chapter 1
Introduction

Nowadays, facilitated by the rise of mobile or web-based technologies, Internet of Things (IoT), social and news media, etc., an increasing amount of data are being generated at a fast rate and large size. Recent statistics show that Internet companies such as Facebook, Google, Baidu and Alibaba generate or process data at the scale of petabytes or terabytes per day [4]. In addition to the explosion of speed and volume, an increasing attention has been drawn to explore the hidden value in big data. They are being used to transform the domains of healthcare, business, manufacturing, engineering, science, etc., for innovation, competition and productivity according to the report by Mickinsey [5]. Those trends have brought about significant challenges and problems regarding how large volume of data can be stored, organized and analyzed in a proper and efficient fashion.

Cloud computing, or simply put as the "cloud", has emerged as the state-of-the-art solution for big data in terms of storage and processing. According to the definition of the *National Institute of Standards and Technology (NIST)*[1], cloud computing is a computing service provisioning model for enabling ubiquitous, on-demand access to a shared pool of resources, including network, storage, computation, etc., that can be configured with rapid elasticity [6]. Cloud computing has revolutionized IT industry by lowering the barriers to large scale computing resources. As a result, enterprises or individual users can benefit from the pay-as-you-go model of resource offering with great flexibility and cost savings.

The advantages of cloud computing have further promoted the rapid development of big data technologies and applications in recent years. A variety of cloud applications have been developed to meet the need of big data processing, storage

[1] http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf

and management. For instance, MapReduce based data processing model has attracted dramatic attention since it was firstly introduced by Google in 2004. The availability of Hadoop, an open source software framework for distributed data storage and computation, leads to wide adoption in both industry and academia for big data applications. Subsequently, the Hadoop ecosystem further expands as a diversity of applications built upon Hadoop system emerge as solutions for better data storage (e.g., HBase), management (e.g., ZooKeeper), querying (e.g., Hive) and processing (e.g., Spark), etc.

In addition, when it comes to big data processing and analytics, machine learning models or deep learning models have become increasingly prevalent and widely applied to process and learn from a large volume of data at scale. With the growing adoption of cloud computing, many cloud applications, e.g., Azure Machine Learning Studio by Microsoft and Tensorflow-based service by Google, started offering online services that allow users to outsource the task of training a machine learning model. Moreover, users can even choose to store and deploy models that are either publicly shared or privately owned, which makes it easy to develop analytical solutions for various real world problems such as text or image classification.

With the proliferation of various cloud services for big data provided by Internet companies such as Google, Facebook, Amazon, Cloudera, etc., more and more data are being moved to the cloud. The migration of data to the cloud inevitably leads to concerns regarding data security and privacy. A myriad of potential threats and risks have been stressed in various works [7–9]. In traditional architectures, data are usually maintained in privately owned servers or data centers, where data owners take the responsibility to install protection measures and oversee the operations on data. In contrast, the cloud is a dynamic and heterogeneous environment where resources are shared. Users have to place the trust on cloud service providers in terms of how their data are protected. Particularly, users no longer have full control over their data whereabouts and their management. In order to fully promote adoption of safe cloud services, it is vital for the service providers to demonstrate that cloud can be trusted as well.

Particularly, in the case of machine learning based data analytics powered by the cloud, machine learning models have excelled in some of the most challenging tasks of analyzing textual or imagery data, which brings about their further

application to many security critical areas such as spam filtering, face recognition based authentication, etc. Such prevalence has drawn attention from malicious attackers as well as security researcher and practitioners to explore the potential vulnerabilities within the learning model itself.

There already exists various types of well known attacks against machine learning models. For instance, a well crafted sample can fool a model from classifying it correctly. A set of malicious samples, injected during the model training process, can implicitly affect the behaviors of the resulting model, which could lead to undesirable consequences of misclassification. Hence, it is of great importance to understand the security implications of such attacks against machine learning models, in order to improve their robustness in adversarial machine learning scenarios.

In this dissertation, we intend to mitigate the aforementioned concerns and promote trust between cloud users and service providers by enabling users with certain controls over the course of data processing and storage. In particular, we focus on investigating the data flow observed when data are are being stored, accessed and processed. Specifically, we dig into data flow of two typical cloud applications, i.e., MapReduce data processing framework and Hadoop distributed File System (HDFS). Such data flow can provide rich information regarding how data are managed and used in the cloud, with which we can assist users in data protection by active monitoring and intervention according to certain policies and constraints specified by users. Therefore, users are placed in a more favorable position to inspect and regulate how their data are handled.

Furthermore, we are also interested in the emerging security issues of adversarial machine learning, and particularly study how machine learning models are subject to the influence of certain attacks to cause misclassification without affecting its efficacy, for the hope of exploring potential defense strategies against them. Specifically, we look at two new types of attacks against machine learning models in different adversarial scenarios. Bearing a similar goal of intentionally causing misclassification to accommodate certain input samples, one targets the learning model directly by manipulating its parameters. The other aims to generate a model which is seemingly normal but injected with certain backdoor that can be exploited later. Such backdoor is realized by injecting malicious crafted samples in the data when the model is trained.

In the following sections, we provide three typical examples of cloud applications

by investigating their relevant security problems, and discuss the motivation behind our work.

1.1 Anomaly Detection in MapReduce

In the era of big data, there is an increasing need to process massive volume of data in a timely fashion. Promoted by Google, MapReduce is a novel data processing tool that allows for processing of massive dataset in a parallel manner. The framework, consists of two simple functions Map and Reduce, is rather simple and easy to use from a programming perspective. In addition, it also support high fault tolerance and scalability. Hence, it has been widely adopted by industry and applied for various data analytical applications.

Despite its appealing benefits, MapReduce computing paradigm is vulnerable to threats posed by third-party attacks due primarily to inadequate authentication and control over the worker nodes [10]. The risk of malicious worker nodes which either cheat on computation tasks or tamper the intermediate or final results is non-null [11]. Since users are placed at an unfavorable position to oversee the computation carried out by individual nodes, such malicious acts can easily go undetected. MapReduce computations can be threatened also by subtle attacks (e.g., code injection) that may compromise user submitted applications, corrupting the integrity of computation and data.

To address the above challenges, there is an increasing amount of work on computation and data integrity of the MapReduce environment. To detect malicious workers and verify the correctness of MapReduce job, common approaches and techniques range from replication scheme [12,13], crypto-based attestation protocols [11], probabilistic verification methods [13, 14]. In order to obtain strong security guarantees, most of the approaches require modification of original MapReduce work flow by adding extra computation due to either cryptographic attestation or repeated MapReduce tasks. Another approach based on log and execution trace analysis is proposed in [15]. Although no additional modification is needed, both Hadoop log files and execution traces grow exponentially as data becomes bigger, thus suffering from analytical inefficiency. Hence, a desirable goal is to develop a solution that is both effective and yet computationally efficient.

1.2 Data Placement Control in HDFS

Cloud-based storage services have expanded exponentially in recent years due to the emergence of cloud computing. The flexible and reliable data storage offered at a low price has attracted more and more individuals and companies to move their data to the cloud, especially as the amount of data being generated grows rapidly in the so-called Big Data era. Despite the benefits of migration of data to the cloud, cloud adopters have voiced issues related to data security and privacy [7, 9]. Their main concerns are about the lack of the control on how their data are managed behind the scene, and where data are physically located. In some cases, organizations who deal with sensitive data (e.g., medical records) are required to place strict control over data location and access by law. For instance, Canadian laws demand that personal identifiable data must be stored in Canada. Storing such sensitive data in the cloud provides no easy guarantee of adherence to regulatory compliance since cloud storage service is usually operated in data centers across different geographic locations. Therefore, there is a clear need for mechanism that allows users to select regions or territories where their data will reside [16] at the time of upload and consistently throughout the storage life cycle.

To mitigate the issue of the lack of data placement control, some strategies [16] have been proposed to help data owners verify the data locations in the cloud, such as proof of data possession (PDP) [17, 18] or proof of retrievability (PoR) protocols [19–21]. However, the existing works have several critical limitations that prevent them from being widely adopted in the real cloud. First, most of previous proposals (e.g [19]) can only handle the scenario when there is a single copy of a file in the cloud. This is however unrealistic in the real cloud setting whereby a file typically has multiple copies to ensure reliability. Second, among the few works [22, 23] that consider the multiple data replicas, they all rely on heavy encryption operations that are computational expensive to the data owners. Last and most importantly, none of the existing works provides a *proactive location control* at the beginning of the file storage. They all just conduct post-checking of file locations, which may already lead to privacy breach if files are not stored at required locations in the first place. Hence, a systematic and more importantly practical solution is desired.

1.3 Attacks against Machine Learning Models in Adversarial Scenarios

Machine learning approaches have been successfully adopted to address various applications for data analytics on large datasets (e.g. spam filtering, image classification). In parallel, with the growing adoption of cloud computing, cloud services have increasingly offered online services to train, store or deploy machine learning models in a simple-to-use manner. For example, Microsoft Azure provides a full suite of machine learning cloud-based services that enable users to train, test, deploy or even share analytic models for classification tasks [24]. Similarly, Google also provides cloud-based tensorflow service [25] that allows users to download/upload or store machine learning models at the server, and even deploy if necessary.

Further, compared to traditional machine learning approaches, deep learning models have demonstrated tremendous advantages and have excelled in a variety of domains such as computer vision (CV) [26], natural language processing (NLP) [27], automatic speech recognition (ASR) [28], etc., with the ability to process and learn from massive amount of data at large scale. The success of deep learning has led to applications in a number of security-critical areas including malware classification [29] and spam filtering [30], face recognition [31] and self-driving vehicles [32].

When machine learning models are deployed and used for security sensitive applications, the robustness of the model is of great importance [33]. However, due to the probabilistic nature of these predictive models, they may be vulnerable to well crafted malicious input in an adversarial environment. Researchers from both security and machine learning communities have investigated the vulnerabilities exposed by various types of attacks, e.g., evasion attack [33] and poisoning attack [34], against machine learning models [35]. The former usually starts with manipulating the input samples by adding certain noises or obfuscating features to baffle the model into misclassifying them. The latter intends to degrade the efficacy of a model by poisoning its training process with malicious samples.

Motivated by these well-known attacks, we take a further effort to study two new types of attacks with a similar goal that a learning model can be misled to misclassify certain malicious input. The first one named as model manipulation

attack, in contrast to evasion attack, manipulates the parameters of the model instead of the input sample. While, the second one, known as backdoor injection attack, tends to inject a backdoor, one used to trigger misclassification later, into the model by means of poisoning the training data with malicious samples. Unlike typical poisoning attack, the poisoned model can still perform well on normal samples.

1.4 Contribution

Regarding the problem in 1.1, we propose a computational provenance system that analyzes provenance data of computational tasks captured within MapReduce framework. It will not only consider high level data flow but also low level execution details, in order to detect various anomalies in MapReduce job that could compromise data and computation integrity.

Specifically, our contributions are as follows: 1) we highlight how provenance information regarding MapReduce computation can be collected at worker nodes in a distributed manner; 2) we specify a simple set of invariants, against which aggregated provenance data are analyzed for anomaly detection; 3) we conduct a set of experiments, which show that our proposed provenance system introduces a small overhead, and is effective in detecting various anomalies during the execution of MapReduce tasks.

In terms of the problem in 1.2, i.e., data placement control, we propose to address it from a system-oriented view. Specifically, we extends HDFS to achieve location-aware data storage in the cloud. Our contributions are as follows: the enhanced HDFS 1) consistently enforces a location-aware data loading and storage by assigning datanodes according to user specified privacy policies; 2) actively tracks and dynamically corrects possible data migration (due to balancing or data replication needs) within the cluster that might violate data placement policies; 3) monitors potentially malicious data migration, by monitoring socket communication between individual datanodes and correlating it with the constraints imposed by the policy.

As to adversarial machine learning problem, discussed in 1.3, we propose two types of attacks against machine learning model in adversarial environment. With regard to model manipulation attack, our contributions are summarized as follows.

First, we introduce a new type of adversarial scenario where machine learning models are subject to potential threats posed by an adversary who has access to the model at the server side (cloud), compared to common attacks in the typical setting of adversarial machine learning. Second, we extensively evaluate the proposed manipulation attack on two representative types of machine learning models, i.e., a linear model Logistic Regression (LR) and a deep learning model Convolutional Neural Networks (CNN), using two general types of dataset, i.e., texts and images.

In terms of backdoor injection attack, our contributions are three-fold and summarized as follows. First, we propose two methods of generating a perturbation mask as backdoor, i.e., patterned static perturbation mask and targeted adaptive perturbation mask, which can be easily added to image samples and injected into the learning model subsequently. Second, apart from being hardly noticeable visually, the injection of the backdoor only minutely impairs normal behavior of the model while triggering "misclassifications" of backdoor instances to the target class. Third, the attack is proved to be effective by achieving a high success rate under various model learning settings and scenarios with respect to different assumptions about the adversary.

1.5 Outline

The rest of the proposal is organized as follows. Chapter 2 presents a provenance-based approach to monitor the data flow of MapReduce computational tasks and detect possible anomalies indicating malicious or cheating worker nodes that could compromise the integrity of data and computation. Chapter 3 demonstrates an approach to extend the existing Hadoop file system to support policy-driven location-aware distributed storage. Chapter 4 and Chapter 5 respectively describe two types of attacks, i.e., model manipulation attack and backdoor injection attack, against machine learning models in adversarial scenarios. Finally, in Chapter 6, we conclude and discuss possible continuing work in the future.

Chapter 2
Detecting Invariant Violation in MapReduce Computational Data Flow

2.1 Introduction

In this chapter, we propose a rich computational provenance system that analyzes provenance of computational tasks captured within MapReduce framework, not only to track users' data and computation, but also to efficiently detect possible invariant violations which may have compromised the computation. Provenance, also referred as lineage or pedigree, is generally defined as the information that helps determine the derivation history of a data product, starting from its original resources [36]. The importance of data provenance in MapReduce has been recognized by previous work [37] for workflow debugging purposes. In this chapter, we take a step further by encapsulating provenance with a richer context from both data and computational perspective. In our case, provenance not only consists of the derived data product (e.g., input, intermediary, output) and operations (e.g., read, write) deriving the data, but also includes attributes of data (e.g., permission) and computation information (e.g., execution time). Such information, when properly stored and analyzed, constitutes crucial evidence to uncover invariant violations that could possibly indicate tampering of data and computation in MapReduce. Our system is designed to capture and analyze the high-level data flow as well as the low level execution details of MapReduce tasks, thus enabling users to supervise the overall

computational process and identify possible invariant violations.

The rest of the chapter is organized as follows. Section 2.2 describes the related work. In Section 2.3, we provide background about Hadoop, and discuss our goal and assumptions in Section 2.4. We present an overview of our approach in Section 2.5. In Section 2.6, we illustrate our computational provenance system functions in details. We discuss experimental evaluation in Section 2.7, and conclude in Section 2.8.

2.2 Related Work

Our work draws from existing work on provenance and security for MapReduce. We discuss the main contributions in this space in what follows.

Provenance System The problem of capturing and managing provenance data for computational tasks in scientific domain is well studied [38–42]. Provenance data can be collected at various levels given the different capturing mechanisms used by the provenance systems. Workflow-based systems e.g., Kepler [43], Taverna [44], rely on the workflow engines to collect provenance information through the execution of the workflow. With respect to host-based provenance, in systems such as *PASS* [40], provenance information is collected at the kernel level of the underlying operating system by keeping track of the system calls, inputs and outputs as the programs were executed on the host. PASS has also been applied to cloud scenarios (e.g., cloud storages [41, 45]). Further, application-based provenance systems, e.g., specifically designed for MapReduce-based systems, have been recently proposed. RAMP [46] uses a wrapper-based approach to support fine-grained data provenance through each phases of a MapReduce workflow, and provided back-tracing capability for MapReduce workflow debugging purposes. HadoopProv [42] adopts an efficient design by enabling provenance tracking in Map and Reduce phases separately, and deferring provenance graph construction by combining Map and Reduce provenance information to query stage.

In general, work on provenance so far has focused on building provenance model to support provenance storage and querying capability. All of these contributions suffer from some performance issues due to significant overhead in provenance capture. In this work, we use a similar provenance tracking mechanism as Hadoop-Prov [42]. However, our approach operates at file level rather than at the record

level, therefore achieving better performance. Further, we collect information about the context wherein computation occurs (where, when, by which nodes, storage units etc), which allows us to provide detailed provenance information not only about the data, but also about its whole process and processing environment. As a result, we support a detailed invariant violation detection mechanism purely based on our collected provenance data and some simple invariants.

In addition, the problem of securing provenance information has received increasing attention over the years [47–50]. Braun et al. [51] argue for separate security model for provenance data among other regular data. Hasan et al. [49] propose a secure collecting and auditing mechanism against tampering of provenance history data. Bates [50], Rosenthal [52], and others [53] proposed various ways to achieve secure and fine-grained access control for managing provenance information. We are complementary to this line of work, both with respect to goal and methodology. The goal of secure provenance is to protect the integrity and confidentiality of the provenance itself from being compromised. Therefore, cryptographic approaches are mostly used. Rather, we intend to uncover invariant violations by gathering provenance data related to the application itself, and do this by processing various types of data and computational information.

MapReduce Security With respect to secure and private computation on sensitive data, Roy et al. [54] present a MapReduce-based system – Airavat that added mandatory access control to local and HDFS files by using SELinux to execute malicious code, and enforced differential privacy to protect aggregated output from leaking private information in the input. Dyer et al. [10] studied the issues of inadequate authentication in MapReduce applications, and conducted a detailed analysis of the relevant threats on MapReduce resources and job executions. Moreover, a growing body of work has focused on integrity issues of MapReduce data processing. Wei et al. [11] developed a novel framework – SecureMR that implements a scalable decentralized replication-based verification scheme to protect the integrity of data processing service based on MapReduce. Xiao et al. [12] proposed an accountable MapReduce system which introduced a audit group cluster to carry out accountability tests by replicating works' tasks to detect malicious nodes. With respect to computation integrity, replication schemes in [11,12] achieve high reliability and accuracy but suffer from large computation overhead. Closer to our work, Yoon et al. [15] compared system call traces of MapReduce program with

Hadoop logs in search of possible invariant violations as well. Although system call tracing analysis does not require additional MapReduce computations, Yoon's approach demonstrates that system call tracing is too fine-grained to be analytically efficient when the magnitude of computation increases. As a result, only a high-level analysis of overall computational pattern is possible. Our approach draws from the idea of analyzing logs, but greatly extends it. By collecting and integrating data from various sources, we obtain rich provenance information about computation with reasonable granularity, without sacrificing the overall performance.

2.3 Background

Hadoop [55] is an open source software that supports large-scale data storage and processing across cluster of commodity machines. It consists of several core modules, i.e., MapReduce, Hadoop File System (HDFS) and YARN.

MapReduce [56] is a programming paradigm that enables parallel large-scale data processing on a cluster of machines. It involves two basic functions: *Map* and *Reduce* Map function takes input data in the form of key/value pairs and produce lists of key/value pairs grouped by the key. Reduce function merge the intermediate results and generate a collection of values based on the same key. In the context of Hadoop, a standard work flow of a MapReduce job's computation starts with the launch of Map tasks, which will take input splits, i.e., data stored in HDFS, and produce intermediate results. Reduce tasks will be created afterwards, to process the outputs of Map tasks, which will go through a shuffle stage where they will be merged and sorted. Lastly, Reduce tasks will write the final results on to HDFS.

Hadoop file system (HDFS) provides distributed data storage for MapReduce applications. A HDFS cluster has a single NameNode and multiple DataNode in a master-slave architecture. NameNode (master node) manages data storage and access by clients. DataNodes (slave nodes) are where user data are actually stored in the form of data blocks. Clients perform actual file I/O with DataNodes directly. HDFS also provides other features such as file permission and authentication, rack awareness, load balancing, data replication, etc., to support scalability, security and privacy.

YARN [57] is a new module in the lastest version of Hadoop. It takes the role of JobTracker in the early versions of Hadoop. However, it assigns the functions

of JobTracker, i.e., resource management, job scheduling and monitoring into two separate parties: *ResourceManager (RM)*and per-application *ApplicationMaster (AM)* . RM manages system resources among all application (i.e., Map and Reduce tasks running on slave nodes) globally. It is also responsible for scheduling tasks on different nodes. On each node, AM together with per-node *NodeManger (NM)* executes and monitor individual tasks.

2.4 Design Goals

Our primary *goal* is two-fold: First, we aim to develop a rich and yet efficient provenance framework for MapReduce computations, focused on the computational process and its integrity, rather than the derived data product. Second, we aim to investigate the extent to which we can uncover invariant violations indicating misbehaving or malicious nodes or applications by analyzing the provenance of computation and data.

We note that provenance here refers to information describing the computational process underlying a MapReduce application. It involves collecting data from entities that carry out the computation as well as their relationships in terms of the data flow. Specifically, we aim to gather, parse and analyze information regarding the whereabout of the users' data, the actions exerted on the data, the nodes responsible for the actions, statistical metrics of the computation process, etc. By analyzing the lineage of the data and of its computation, we aim to obtain investigative evidence to detect anomalous actions occurred during the computation, and determine their cause of failure.

Note also that we do not intend to detect faulty nodes that could produce incorrect results due to system problems, although this information may become available as a result of our invariant violation analysis. Fault diagnosis in MapReduce-based has been explored by previous work [58]. Further, we also do not attempt to uncover all sorts of invariant violations in a system, but focus on the faults related to MapReduce computation process. Finally, we do not aim to detect compromised user submitted code that could maliciously corrupt the system by exploiting potential vulnerabilities in the underlying OS or JVM [10]. This is relevant to malware analysis and detection, but outside our research scope.

Our approach toward addressing the above goal relies on the following *assump-*

tions.

- All machines in the cluster support our modified version of Hadoop with Yarn framework.
- Master node and major Hadoop components (HDFS, YARN) are trusted. The master node is a crucial element of the Hadoop cluster wherein HDFS NameNode manages data and handle interactions with clients, and ResourceManager arbitrates system resources, schedule and monitor tasks with ApplicationMaster. We also assume MapReduce framework is not tampered with since it provides APIs that enable users to build Map and Reduce functions as intended.
- The Map function executed by the nodes is deterministic: that is to say, given the same input, MapReduce yields the same output.
- There is no guarantee of correctness of MapReduce code submitted by end users through web console or management interface. Since the behavior of workers largely depend on the Map and Reduce functions defined in application code, a compromised user application leading to misbehaving or malicious workers is the main threat considered in this work.

2.5 System Overview

We build a provenance logging system for computational tasks in Hadoop. In our design, we consider computational provenance as an intrinsic component in MapReduce framework. In other words, provenance logging is carried out along with the execution of MapReduce tasks without extra or external assistance. Accordingly, we devise an approach that extends the latest version of Hadoop by integrating provenance logging mechanism within MapReduce framework. The regular logging mechanism in Hadoop is component-centric, i.e., each major entity (NameNode, DataNode, NodeManager) maintains its own log of execution details, some of which are relevant to MapReduce tasks. Although there exist system logs for each task of a user job, they fail to provide a holistic view of the execution of a job. As a result, computational and data lineage are extremely hard to track. For example, the task logs on a worker node only reveal the computational traces and patterns of tasks

executed by that particular node. The behavior of a malicious worker who skips some computation can go unnoticed if only an individual task log or tasks log from a single node are referred.

Although existing provenance tracking techniques [42] manage to capture data lineage at record level, they provide no insight into how a record of data is processed. Here, we argue for a MapReduce-centric logging approach that organizes provenance information in an integrated fashion with both high-level and low level details. Throughout the life cycle of a MapReduce job, our provenance logging system keeps track of the launch of individual tasks in both Map and Reduce stage, and their relationships regarding how data flow through them. We collect data based on the following events/actions: 1) input and output, e.g., we validate the data sources are valid and correct according to user configuration, 2) low level operations, e.g., we keep track of read and write operations on slave nodes, 3) network activities, e.g., we monitor data transmission over network between data nodes, 4) integrity of intermediate results, e.g., we verify output by Map tasks match inputs of Reduce tasks, 5) access control, e.g., we check permission information regarding data and directories in HDFS, 6) execution time pattern, i.e., we monitor the execution time of all tasks. Such information, gathered from each worker node, is constructed in a temporal order to inform a coarse-grained perspective of data flow as a MapReduce job is being processed by Hadoop. Additionally, for each individual Map and Reduce task, we also track low level details of computation, e.g., read and write activities, task execution time, etc., to provide a fine-grained view of how individual tasks perform with the corresponding data.

The provenance profile built with our method not only specifies the overall sequential steps of data processing of a MapReduce job, therefore providing information about the data being produced by the application. It also serves as critical investigative evidence providing insights into detecting various invariant violations. The invariant violations detected by our provenance system are discovered by analyzing and comparing provenance information with expected invariants, which should hold true regardless of the specific application and input file being processed. They can be classified into four types:

1. Deviation from predefined configuration of jobs submitted by users,

2. Inconsistency with respect to intermediate results,

3. Suspicious operations during the execution of users' jobs,

4. Compromised computation of individual tasks.

Intuitively, the first two types of invariant violations indicate the existence of possibly faulty or compromised nodes in some way. As for the third class, typical instances of suspicious activities could be unauthorized access (e.g., read operations) to users' data or intentional access (e.g., write operations) to publicly accessible directories in HDFS etc.

2.6 Design and Implementation

Our computational provenance logging system consists of three components: *provenance capturer, log parser and invariant analyzer* (see Figure 2.1). *Provenance capturer* generates raw logs containing data and computation related information, which are further turned into a provenance profile by *log parser*. The *invariant analyzer* analyses the provenance profile to identify possible invariant violations.

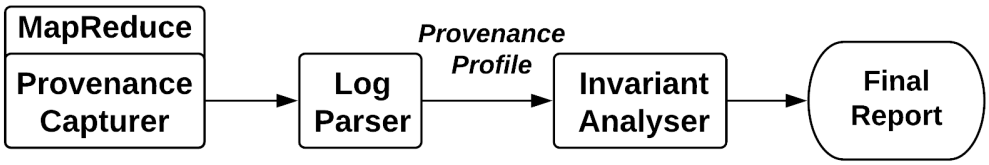

Figure 2.1: Provenance logging system.

2.6.1 Provenance Capturer

In order to build an efficient and reliable provenance capturing mechanism, we choose to embed the logging function in MapReduce framework, and specifically extend provenance logging to every phase of execution of a MapReduce job. The whole provenance capturing process during the execution of a MapReduce job is illustrated in Figure 2.2.

The latest version of Hadoop adopts YARN framework, which supports a per-application ApplicationMaster (AM). The AM's main responsibility is to request

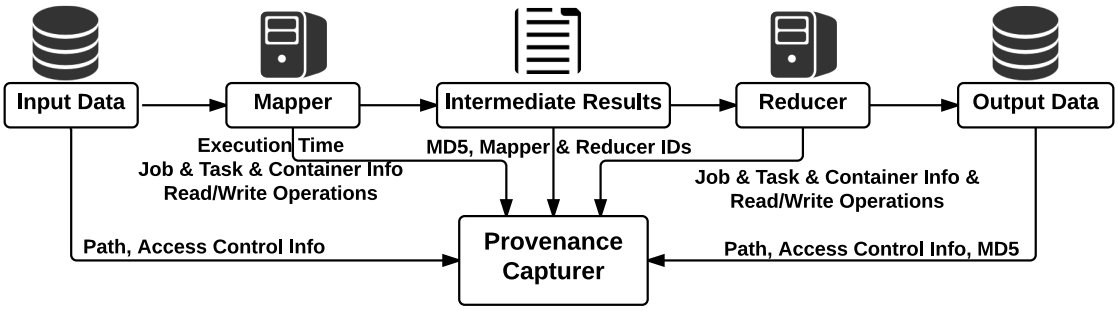

Figure 2.2: Provenance capturing process.

resource containers in which individual Map and Reduce tasks are executed, and track their status and progress. Therefore, the initialization of resource containers marks the launch of either Map or Reduce tasks. At the beginning of Map and Reduce phases, the provenance capturer at the worker nodes log information about resource container and corresponding tasks.

We introduce six types of provenance entries in the raw log files which cover both high and low level execution details of Map and Reduce tasks as well as the corresponding relationship between different entities (i.e., container, Mapper, Reducer) identified by unique IDs.

The script shown below is an example of a container and a task entry. Container entry indicates the correlation between container and task. In addition to task_id , task entry also includes profile of a job (i.e., name, id, user who submits the job) that the task belongs to.

```
[CONTAINER]Assigned container container_1403624155095_0001_01_000002 at
isthadoop01.ist.psu.edu:8042 to task attempt_1403624155095_0001_m_000000_0
on node isthadoop01.ist.psu.edu:54830
[TASK]User:  hduser Job:word count JobID: job_1403624155095_0001 TaskID:
attempt_1403624155095_0001_m_000000_0
```

During Map phase, we track the source of input of Map tasks. Specifically, we log the path of input data in Hadoop file system and its access control information to check whether the input are valid or not. An example of input entry for Map tasks follows:

```
[INPUT]Mapper attempt_1403624155095_0001_m_000000_0 has input:  hdfs:
//isthadoop02:9000/user/hduser/test/pg5000.txt:0+1423803 Owner:  hduser
Permission:  rw-r-r-
```

Output values of Map tasks are equally important, since they later become the

input of Reduce tasks. Hence, for output entry, we track the message digest (MD5) of intermediate output for integrity checking as the instance shown below.

```
[OUTPUT]Output by Mapper attempt_1403624155095_0001_m_000000_0 MD5:
71dee21483900cf135b3bb2689b4ed84 Length:466823
```

During Reduce phase, Reduce tasks will take the output of Map tasks as input and produce the final results as output. Therefore, the validity of input and output of Reduce tasks are essential to the overall computation. We track where the input data are coming from and whether they are valid. For output of Reduce tasks, we log the final locations with the corresponding access control information, and message digest of output data. Examples of input and output entry for Reduce tasks are as follows.

```
[INPUT]Reducer attempt_1403624155095_0001_r_000000_0 get input from
Mappers:  attempt_1403624155095_0001_m_000000_0
[INPUT]Reducer fetched input from Mapper attempt_1403624155095_0001_m_000000_0
MD5:  71dee21483900cf135b3bb2689b4ed84 Length:466823
[OUTPUT]Final output by Reducer attempt_1403624155095_0001_r_000000_0
to hdfs://isthadoop01:9000/user/hduser/output MD5:
d2f132850550a64240510fe06b1379b7 Length:  337648 Owner:  hduser
Permission:  rwxr-xr-x
```

All the provenance entries mentioned above provide a high-level overview of the computation carried out by Map and Reduce tasks. To complement this high-level perspective, we also keep track of low level execution data, from the standpoint of the data stored in Hadoop file system. Hence, during the execution of Map and Reduce tasks, we log read and write operations that access data in HDFS. For instance,

```
[READ]Task task_attempt_1403624155095_0001_m_000000_0 at
/130.203.136.232:50010 read from /130.203.136.232:56583 with data block
BP-1211802040-130.203.136.30-1393906711358:
blk_1073742599_1776 at time 1403638967086
[WRITE]Task task_attempt_1403624155095_0001_r_00000_0 at
/130.203.136.233:33427 write to /130.203.136.232:50010 with data block
BP-1211802040-130.203.136.30-1393906711358:
blk_1073742604_1781 at time 1403639009380
```

From the script above, the system tracked that a Mapper (min task ID means Mapper) with ID task_attempt_1403624155095_0001_m_000000_0 on the host

130.203.136.232 requested a data block blk_1073742599_1776 from a datanode with IP 130.203.136.232 at the time of 1403638967086 (in milliseconds), and a Reducer (r in task ID means Reducer) with ID task_attempt_1403624155095_0001_r_0000_0 on the host 130.203.136.233 saved a data block blk_1073742604_1781 on a datanode with IP 130.203.136.232 at the time of 1403639009380 (in milliseconds).

2.6.2 Log Parser

Upon completion of a MapReduce job, the generated raw provenance log files are transmitted via secure channels using SSH from multiple slave nodes to master node where the log parser will organize every piece of information and produce a single provenance profile. The provenance profile reports summary execution information about the MapReduce job, encoded in XML. A template of provenance profile is presented in Figure 2.3

The root element is the MapReduce job. From the task entries, the profile tracks the job attributes (i.e., id, name, user) and the task IDs which uniquely identify the tasks that belong to the job. The job element has several children elements named task . Within each task element, we have input, output, read and write as sub elements. For each Map and Reduce task, we can extract its input and output related information from input and output entries, and details about read and write operations by the task from read and write entries.

Specifically, for Mapper input, we extract and profile the source of input data (path in Hadoop file system) and its corresponding access control information (directory owner and permission). In terms of Mapper output, we are interested in the MD5 value which helps verify the integrity of intermediate results during computation.

As to the input of Reduce tasks, the profile stores the IDs of Map tasks that Reduce tasks are getting intermediate results from and the MD5 values of those input. For Reducer output, the output path in Hadoop file system is stored, as well as the related access control information as well (directory owner and permission).

Since data in Hadoop file system is stored in a distributed fashion, read and write operations would happen over network as one node is getting data from another node. For read and write element,the profile tracks task id , src addr , dst addr and data block id from read an write entries that capture the data

```
<Job ID="" Name="" User="">
 <Config>
   <tmpdir></tmpdir>
 </Config>
 <Parameter>
  <Input_Path></Input_Path>
  <Output_Path></Output_Path>
  <Num_Map></Num_Map>
  <Num_Reduce></Num_Reduce>
 </Parameter>
 <Container ID="">
  <TaskID></TaskID>
  <location></location>
 </Container>
 <Task ID="" Type="", Time="">
  <Input>
   <Perm></Perm>
   <Path></Path>
   <TaskID></TaskID>
   <md5></md5>
  </Input>
  <Output>
   <Perm></Perm>
   <Path></Path>
   <TaskID></TaskID>
   <md5></md5>
  </Output>
  <Read>
   <src_ip></src_ip>
   <dst_ip></dst_ip>
   <data_blk_id></data_blk_id>
  </Read>
  <Write>
   <src_ip></src_ip>
   <dst_ip></dst_ip>
   <data_blk_id></data_blk_id>
  </Write>
 </Task>
</Job>
```

Figure 2.3: Provenance profile template.

transmissions initiated by tasks.

2.6.3 Invariant Analyzer

In addition to producing the standard provenance file of a complete job, our system analyzes the gathered information in search of possible invariant violations upon completion (or interruption) of a MapReduce job. We specifically verify whether the following properties are preserved: 1) input and output paths are valid, according to the user specification; 2) the integrity of intermediate Mapper output is maintained; 3) input and output data are only accessible to authorized users; 4) only the assigned data can be retrieved by worker nodes via network; 5) similar computation pattern (e.g., execution time) is observed among Mappers. We discuss some of the behaviors leading to a failure of the above properties, and the invariants used to detect them, in what follows:

I/O Discrepancies When submitting a Hadoop job, users would provide the path of input data and output destination along with the program. A compromised program would allow an attacker to create malicious MapReduce applications, which intentionally take wrong input data or write final results to different locations during the execution of the codes. In such cases, the whole computation is still compromised even if completed successfully. Comparing the de facto execution information regarding input and output in provenance profile with the known parameters configured by users can easily reveal such discrepancies. On the provenance profile (see Fig. 3), we check whether the following values match:

- the value of <Input_Path> sub-element in <Parameter> with the value of <Path> sub-element in <Task> with attribute Type="Map"
- the value of <Output_Path> sub-element in <Parameter> with the value of <Path> sub-element in <Task> with attribute type="Reduce"

Note that these comparisons are sufficient because users application parameters of the job are saved and written to the provenance profile under <Parameter> element (see Figure 2.3). Upon job completion, the final provenance profile logs the actual number of initiated tasks along with their input and output information (in the <input_path> etc), which is compared against the users' setup.

Access violations HDFS implements a similar permission model as Linux system, where files and directories are associated with an owner and a group. Normally,

users' submitted job should only access the users' own files and directories during the computation if relevant permissions are properly configured. Any failure of permission checking by Hadoop during the computation would lead to the corruption of MapReduce jobs. However, attackers can still exploit HDFS by creating publicly available directories accessed by seemingly benign but corrupted user jobs. For instance, a MapReduce job can be maliciously configured to write output to a public folder set up by attackers without failing the permission checking. Hence, we examine the permission information of input and output directories in the provenance profile to see if they are properly set. This information is recorded in the <Perm> sub-element within all <Task> elements.

Misplaced intermediate Mapper output This type of invariant violation may occur if, for instance, the location of temporary directory where intermediate Mapper output is stored is modified within a compromised MapReduce program using Hadoop provided APIs used to access configuration files. Our provenance profile includes both the location where Mappers write output to and Reducers get input from, from which we can easily detect if there is any inconsistency with the original configuration when Hadoop was installed. Here, we check whether the value of <tmpdir> sub-element in <Config> element matches with the value of <Path> sub-element in <Task> element with attribute Type="Map".

Corrupted intermediate Mapper outputs To achieve the correctness of computation, it is necessary to ensure the integrity of data. In Hadoop, checksums are used to make sure data are not corrupted during transmission over network between data nodes. But it is also essential that intermediate Mapper outputs temporarily stored on local disc are intact. Any of the intermediate results, if compromised by attackers before being fetched by Reducers, can lead to computation failure because of incorrect results. Our provenance profile logs the MD5 value of Mapper output and Reducer input: any inconsistency between them would imply compromised intermediate results. With respect to the Provenance profile we check the value of <md5> sub-element in <Output> element of Map task with the value of <md5> sub-element in <Input> element of corresponding Reduce task.

Skipped computation There could be cases where some of the computation tasks may be skipped by altering certain execution of Map and Reduce tasks of a job. Recent study on fault analysis in MapReduce application [58] has shown that it is reasonable to expect slave nodes to have similar workloads. To cope with this, the Profile logs the time taken to finish individual tasks among all nodes and uncover anomalous time patterns based on comparative analysis of the reported time. Note that, compared to other anomalous behaviors, this detection method is non-deterministic, in that a different usage pattern may not be a definite indicator of an attack, unless the comparison is carried out among hosts that share the same conditions and input type.

Other controls can be performed to verify lower-level information regarding the computational process. For instance, the Profile can be configured to check if a Mapper only retrieves the assigned data by referring to the value of <data_blk_id> sub-element in <Read>element. That is, given a data block ID, we leverage the HDFS command hadoop fsck / -files -blocks | grep <data_blk_id> , to find the file associated with the data block and check if the data block belongs to the input file assigned to the Mapper. Moreover, it is straightforward to verify whether the original configuration set by the user was preserved, by matching the value of <Num_Maps> & <Num_Reducers> with the number of Map Task & Reduce Task elements If any of the invariants is violated, computational provenance is violated, and it is likely that the privacy of user data or computation may be compromised.

2.7 Experimental Evaluation

We set up a small cluster of three machines with two Intel(R) Xeon(R) X5550 2.67 GHz CPUs, 48GiB system memory and Ubuntu 12.04 LTS linux OS. We deploy the modified version of Hadoop on the cluster. In the cluster environment, the provenance logging system will keep track of all the Map and Reduce tasks launched on each slave machine and send generated raw log files as to master node machine for analysis presented in Figure 2.4.

In terms of the Hadoop software, our hosts support an extended version of Hadoop 2.3.0 which adopts YARN architecture. The extension includes our MapReduce framework equipped with the provenance logging as the system described in Section 2.6. Additionally, log parser and analyzer are external components with

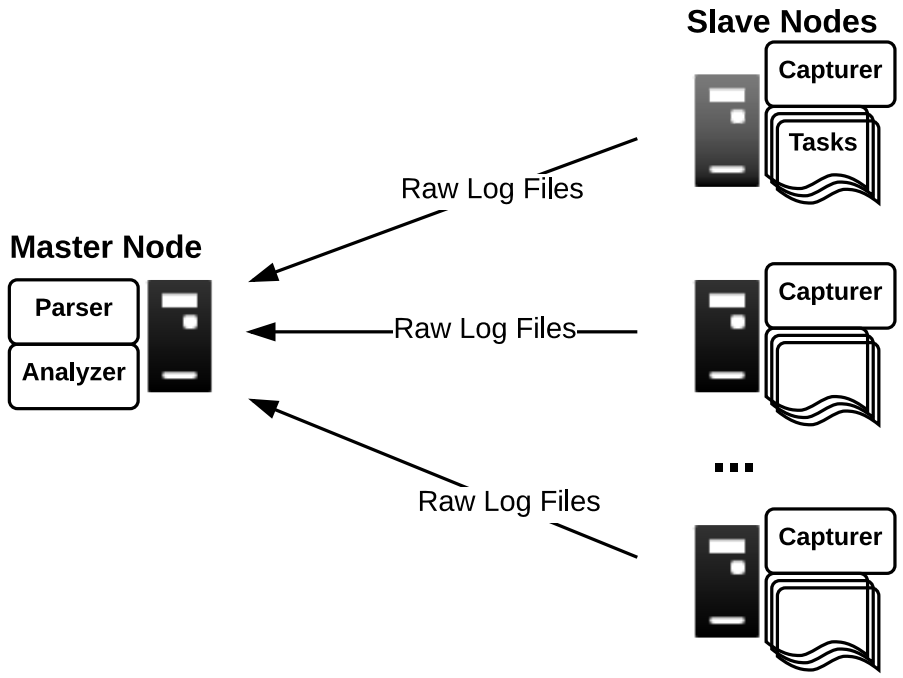

Figure 2.4: Provenance logging system in a cluster.

regard to the logging system in Hadoop. We use the classic Wordcount application for our experiments.

2.7.1 Performance

A design goal of our provenance logging system is to introduce as less overhead as possible while keeping track of provenance information pertinent to the execution of a MapReduce application. We execute the Wordcount application on various size of dataset using the default Hadoop 2.3.0 version, and compare its performance with our extended Hadoop based on the default version. The dataset we use is an ebook (with size of 1.4 MB) downloaded from Project Gutenberg. We replicate the content of original ebook to create new files with different size (e.g., 1.1GB, 5.5GB, 11GB) for the experiment. With each version of Hadoop, we run separate jobs of the Wordcount applications with the input being one of these newly created files and record the total execution time of each job. The results are shown in Figure 2.5.

As we can see, the total execution time of a job increases minimally with the input file size increase for both versions of Hadoop. In the Wordcount application,

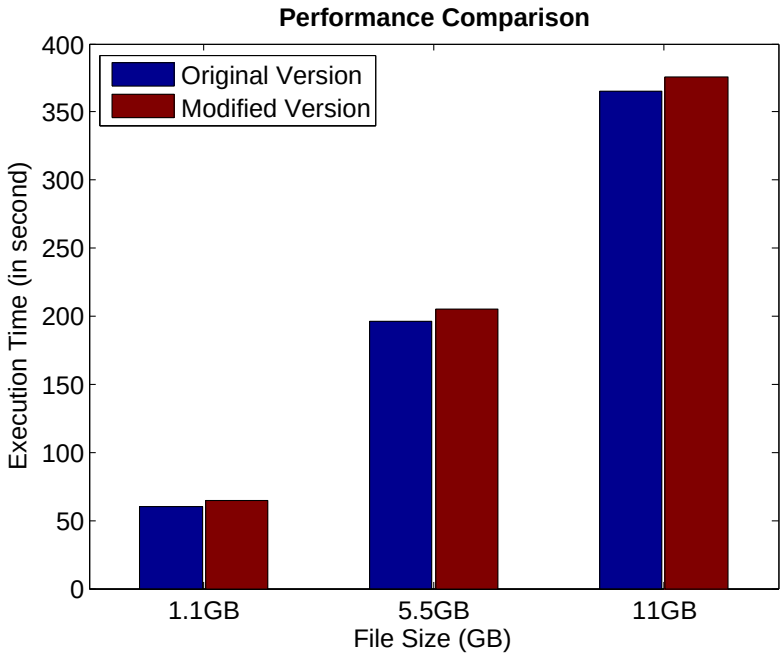

Figure 2.5: Hadoop Yarn versus our provenance-rich Hadoop.

a larger input file implies more words to be processed in Map phase and larger intermediate outputs for Reduce phase leading to increased execution time. Given an individual input, the difference in terms of total execution time between the original and modified version of Hadoop increases slightly as file size increases and is 5, 9 and 11 seconds respectively. Accordingly, the ratio of computation imposed by our provenance system amounts to about 8%, 5% and 3% of the overall computational effort.

2.7.2 invariant violation Detection Results

We describe the setup and report the results obtained after evaluating the anomalous scenarios presented in Section 2.6.3, based on the Wordcount application. The dataset used as input files are two ebooks from Project Gutenberg.

In the first experiment, we explore the anomalous scenario of input/output discrepancies. Corrupting the input and output paths with different locations in user's own directories seems one plausible way to simply compromise the computation itself without failing the permission policy. An alternative approach is set up directories that are essentially open to anyone. Writing the output to the openly

accessible directory can compromise the privacy of user data.

Note that, corruption of the specified input or output might occur since we make no assumption about the correctness of user's MapReduce programs. User submitted Java code could be compromised by injecting pieces of Java code into compiled classes and methods at runtime (e.g., AspectJ [59], Javassist [60]). Users' programs might also suffer from man-in-the-middle attack targeting Jar files containing source codes during network transmission when user submits it or worker nodes retrieve it from master node.

To demonstrate, we provide a proof-of-concept attack by creating a compromised Wordcount Java program which takes additional input file different from the specified one, and write to a secondary location instead. Specifically, we modify the original Wordcount application with regard to setting input, output path and configuration file, i.e., we set

```
FileInputFormat.setInputPaths(job, "original-input-path, additional-input-path")
FileOutputFormat.SetOutputPath(job, "different-output-path")
```

In HDFS, we set up two directories owned by different user accounts:

```
drwx--- cxl491 supergroup /user/cxl491
drwxrwxrwx cxl492 supergroup /user/cxl492
```

The directory /user/cxl491 is only accessible by user cxl491 while the directory /user/cxl492 is open to anyone given the permission setting.

When a user job along with the program is submitted, the logging mechanism keeps track of the specified parameters of input and output. Once the computation is completed, provenance file from each machine is aggregated into a single provenance file, part of which is shown in Figure 2.6.

As we can see, the job gathers data from unexpected input ("pg.4300.txt" & "pg.20417.txt") at a directory different from the specified one, and writes to an unexpected output location ("/user/cxl492/output" , rather than /user/cxl491/output). Thus, the final provenance file reveals the discrepancies between user-specified and de facto input and output paths. It is also easy to observe an invariant violation in that the job takes additional input from and write output to openly accessible directories indicated by perm="drwxrwxrwx".

In the second experiment, we explore the cases of corrupted temporary directory as well as intermediate output (cases c) and d)). To show the possible invariant violation, we create a malicious Java program that modifies the location of temporary

```
<Job ID="1409946794632_0009" Name="word count">
 <Parameter>
  <Input>/user/cxl491/input/pg.5000.txt</Input>
  <Output>/user/cxl491/output</Output>
 </Parameter>
 <Task ID="m_000001_0" Type="Map">
  <Input perm="drwx------">
  /user/cxl491/input/pg.5000.txt</Input>
 </Task>
 <Task ID="m_000000_0" Type="Map">
  <Input perm="drwxrwxrwx">
  /user/cxl492/input/pg.4300.txt</Input>
 </Task>
 <Task ID="m_000002_0" Type="Map">
  <Input perm="drwxrwxrwx">
  /user/cxl492/input/pg.20417.txt</Input>
 </Task>
 <Task ID="r_000000_0" Type="Reduce">
  <Output perm="drwxrwxrwx">
  /user/cxl492/output</Output>
 </Task>
</Job>
```

Figure 2.6: Partial provenance profile of Wordcount in the first experiment.

directory set in configuration file, i.e., conf.set("tmpdir", different-location, "core-site.xml") . As shown in Figure 2.7, the temporary directory was changed from /tmp to /home/cxl491/tmp . In addition, to demonstrate a proof of concept that compromised intermediate output can be captured, we intentionally corrupt the output of Map tasks before being retrieved by Reduce task. In the final provenance profile, we can observe the mismatch between the message digests of Mapper output and Reducer input as shown in Figure 2.7.

In the third experiment, we explore the scenario of cheating Mapper (case e), skipped computation). Specifically, we have six identical files with each being processed by the same amount of Mappers (since a file might be processed by multiple Mappers given its size). Five of them will be processed by benign Mapper(s) that carry out the computation faithfully, and one will be processed by malicious Mapper(s) which intentionally skips some of the computations in word counting. Moreover, we create four different scenarios where the malicious Mappers will skip 20%, 40%, 60% and 80% of the total computation respectively. For each scenario, we run separate jobs of Wordcount application with files of different size (86.9MB,

```
<Task ID="m_000000_0" Type="Map">
 <Config>
  <tmpdir>/tmp</tmpdir>
 </Config>
 <Output>
  <TaskID>m_000000_0</TaskID>
  <md5>71dee21483900cf135b3bb2689b4ed84</md5>
  <Path>/home/cxl491/tmp</Path>
 </Output>
</Task>
<Task ID="r_000000_0" Type="Reduce">
 <Input>
   <TaskID>m_000000_0</TaskID>
   <md5>0239cd4e72bd35588cd80c646fcb4c8a</md5>
 </Input>
</Task>
```

Figure 2.7: Partial provenance profile of Wordcount in the second experiment.

293.3MB, 1.1GB) individually. For each file, we log the execution time by either cheating Mapper(s) or normal Mapper(s). We show the trend of execution time by the cheating Mapper with respect to percentage of skipped computation in Figure 2.8.

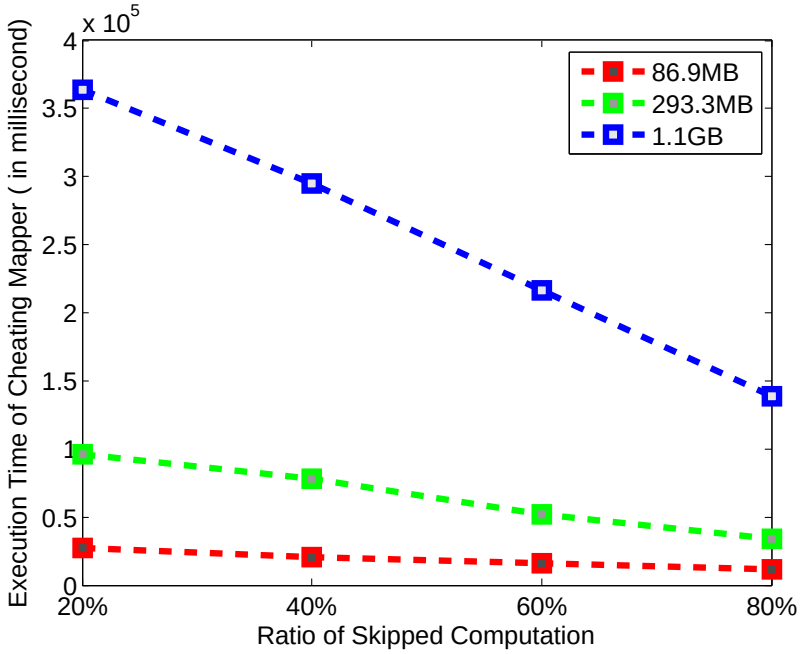

Figure 2.8: Trend of execution time w.r.t. ratio of skipped computation.

In each scenario of skipped computation, for each of the five files processed by normal Mapper(s), we compute the ratio between successful and skipped computation:

$$Ratio = \frac{T_{normal} - T_{cheating}}{T_{normal}} \tag{2.1}$$

where T_{normal} denotes the execution time of normal Mapper(s), and $T_{cheating}$ means the execution time of cheating Mapper(s). The average value of the yielded ratios is shown in Figure 2.9.

Clearly, given an input file, the more computation a cheating Mapper skips, the larger the time disparity with respect to normal Mappers. Such trend is consistent across different input sizes, which means this method may possibly be used to detect cheating Mappers. It can be (roughly) modeled by linear regression in the form of $y = a \cdot x + b$ (with b=0.1) for Wordcount application. Given a ground truth and an observed execution time, we can easily deduce the estimated value of skipped computation, which indicates, in some sense, the likelihood of the existence of cheating Mapper.

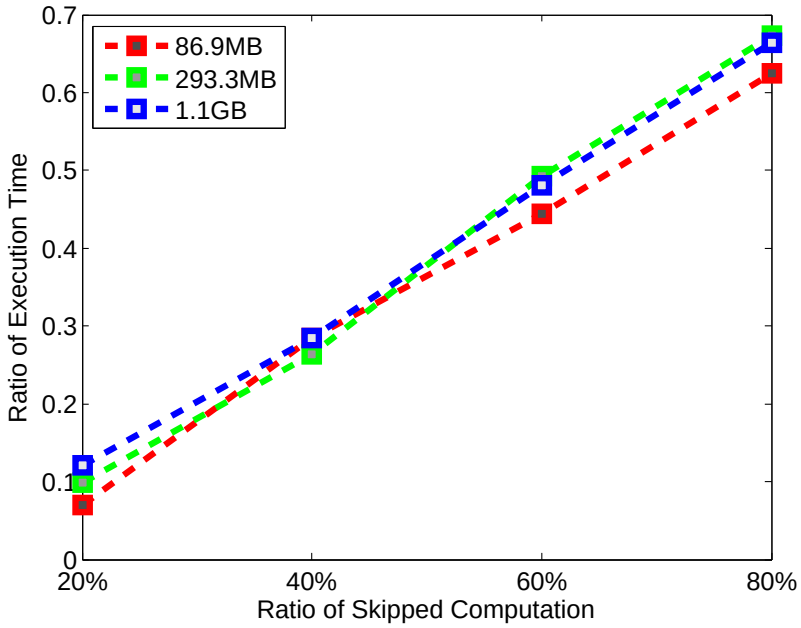

Figure 2.9: Trend of average ratio w.r.t. percentage of skipped computation.

2.8 Discussion and Conclusion

In this chapter, we presented a provenance logging system for computational MapReduce tasks in Hadoop, and proposed a provenance-based approach to detect possible invariant violations indicating malicious or cheating worker nodes. Our experiments showed that our approach gathers provenance information with reasonable level of granularity incurs in an extremely small amount of computation overhead. The collected provenance information can be effectively utilized not only for provenance purposes, but also to detect suspicious behaviors of worker nodes or invariant violations induced by compromised MapReduce programs.

This work is yet an initial attempt to leverage provenance for invariant violation detection in MapReduce. We identified a few limitations, and discuss avenues for further research next.

A first limitation is that the tampering of the input data assigned to malicious Mappers can compromise the final application outcome. Beyond the existing replication approaches [11,12], a more interesting solution would be to combine host-based with application-based provenance. Besides application-based provenance collected within MapReduce framework, the provenance logging system may incorporate host-based provenance gathered at local worker machine with appropriate granularity, which may provide insights into the integrity of input of Map tasks.

Second, the risk that one could disable or tamper with our provenance system is currently non-null. Our current design relies on assumptions of correctness of the Hadoop infrastructure, and the availability of our modified Hadoop version at each worker node. If Hadoop itself or the system at the worker node is corrupted, the logging function would fail. In such case, if the master node is not getting response regarding provenance logs from some worker node, an anomalous behavior in the system is currently detected, indicating a malicious or compromised worker node. Provenance logs might also be tampered and forged. To mitigate, simple cryptographic primitives or existing techniques to secure provenance [49] can be used.

Finally, the invariant violations discussed in this chapter are purposely limited to those that could be indicators of compromised computation or user data. Other anomalous behaviors such as Map or Reduce task failure due to faults or errors of local machines are beyond the scope of our investigation. Currently, these types of

faults would be detected only if they violate any of the invariants that we use for our analysis. However, our provenance system can be easily extended by integrating information gathered from *ResourceManager* in Hadoop, which would restart tasks on failure based on the node status tracked by *NodeManager*

In the near future, we will develop a hybrid approach that not only collects but also protects provenance data in a private and secure way. Furthermore, our statistical approach of comparing execution time patterns between multiple Map tasks still requires thorough investigations under various MapReduce applications. Finally, we will explore the extent to which the provenance system can be refined to deal with scenarios where multiple workers are compromised or colluding.

Chapter 3
Location Policy Enforcement in HDFS Storage Data Flow

3.1 Introduction

In this chapter, we propose a novel systematic approach that achieves proactive location control for files stored in the cloud that has Hadoop File System (HDFS) as the underlying architecture. Our proposed system is called LAST-HDFS (Location-Aware Storage Technique in HDFS), which seamlessly integrates a new set of components into the HDFS to enhance its ability to: (i) consistently enforce location-aware data loading and storage by assigning datanodes according to user specified privacy policies; (ii) actively track and dynamically correct possible data migration (due to balancing or data replication needs) within the cluster that might violate data placement policies; (iii) audit potentially malicious data migration, by monitoring socket communication between individual datanodes and correlating it with the constraints imposed by the policy.

Our proposed LAST-HDFS has two main advantages. First, LAST-HDFS releases users' computational burden on doing location verification and hence revives the original advantage of cloud adoption of saving computing cost. Using LAST-HDFS, users only need to tell the cloud service providers their privacy requirements and the remaining will be handled automatically by the cloud. Second, LAST-HDFS provides a practical tool for existing cloud service providers to honor their customers' privacy concerns. Our proposed new components can be installed as plug-ins to HDFS which will ease adoption in the real cloud.

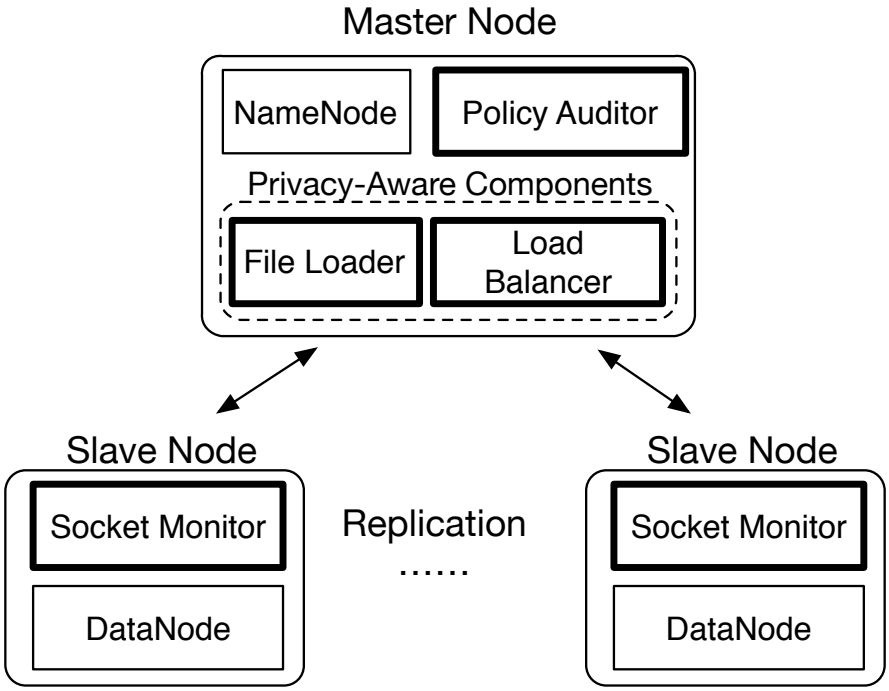

Figure 3.1: Overview of LAST-HDFS system.

The rest of the Chapter is organized as follows. Section 3.2 provides background on write mechanism and load balancing in HDFS. Section 3.3 introduces the design goals, threat model and general design of the system. Section 3.4 presents the implementation details of the proposed LAST-HDFS system. Section 3.5 reports the experimental results. Section 3.6 reviews the related work on location-aware cloud storage. Finally, Section 3.7 concludes the chapter.

3.2 Background

In this section, we provide some background information on the Hadoop Distributed File System (HDFS), which is the base of our system. HDFS [61] is an essential component of the open source Hadoop software. HDFS is a distributed file system designed to run on commodity hardware and support distributed data storage and access by applications running on top of it with high throughput and fault-tolerance. It adopts a master-slave architecture, which consists of a single namenode as the master node and multiple datanodes as the slave nodes. The namenode manages file system meta-data and orchestrates file accesses. The datanodes serve read/write

requests issued by clients and perform the actual read/write operations on disk blocks as instructed by the namenode. In what follows, we briefly review the data storage and load balancing mechanisms adopted by the current HDFS since our proposed system will revise these two functions to achieve location-aware storage.

3.2.1 Write Mechanism in HDFS

For a data owner (client) to upload a file to HDFS, he needs to first initiate a write request to the namenode asking to create a new file in HDFS. Once the namenode approves the request, the client will begin writing data to the stream where data is split into packets. Each packet represents a data block of the file that will be written to the datanodes. A separate thread in the client will pick up a packet and contact the namenode, from which a list of candidate datanodes will be returned to the client. Then, the client will send write packet to the first datanode in the list where the data block will be stored. Subsequently, the data block will be replicated to the following datanodes in the list in a pipeline manner.

3.2.2 Load Balancing in HDFS

Load balancing is of great importance to the overall performance of HDFS clusters, especially when a new datanode is added to the cluster or the disk space of certain datanodes is saturated. Hadoop provides a balancer tool that allows a cloud administrator to balance the disk space usage in a HDFS cluster. An outline of the load balancing process is described below:

1. The balancer partitions all the datanodes into two groups: i) under-utilized node group and (ii) over-utilized node group, based on their data block usage reports.

2. The balancer randomly select one datanode from each group to form a pair of nodes whose load will be balanced by transferring certain amount of data from one to the other.

3. The balancer randomly selects a list of data blocks in the over-utilized datanode and transfer the data to the under-utilized datanode in the same pair.

4. The balancer iterates the above three steps until all the datanodes in the cluster reach certain utilization threshold, i.e., the system achieves balanced load.

3.3 System Overview

3.3.1 Design Goal

Our overarching goal is to enable HDFS to support location-aware data storage so that data owners' location privacy policies are strongly enforced when storing their data in the cloud. Recall that in the existing HDFS, the locations of a user uploaded file are determined by two factors: i) data replication for the purpose of fault-tolerance, and ii) load balancing to optimize cluster space utilization. In other words, users' file blocks will be replicated to multiple datanodes when the files are uploaded for the first time, and it is very likely that the file blocks on saturated nodes may be transferred to under-utilized nodes at a later time.Accordingly, we need to achieve the following goals:

1. When uploading the files to the cloud, users should be allowed to specify the location constraints (e.g., regions, countries) within which their data is allowed to be placed in the cloud.

2. The location constraints (i.e., location privacy policies) specified by the users should be consistently enforced during the data replication process.

3. The location constraints should also be consistently enforced during the load balancing process.

4. Any data movement within the file system that violates the location constraints should be detected.

3.3.2 Threat Model

A typical data storage operation in HDFS involves three main components: namenode, datanode and file loader. The namenode is a master node in HDFS which manages the entire file system and also interacts with users. The datanodes store

the user data and perform the read/write operations on files as instructed by the namenode. The file loader is in charge of uploading files to the cloud on behalf of users.

In our work, we assume that the namenode is fully trusted because it is the core node in the system which should be well protected. With that said, the namenode will faithfully handle requests from users. However, given that a cluster usually includes a large number of datanodes, leading to a higher chance that some of them may be compromised by attackers, we *do not* assume all the datanodes are fully trusted. The compromised datanodes could intentionally transfer or copy users' data to any other nodes that may reside outside the legal regions specified by the users. Attackers may do this for various purposes such as analyzing users' data for advertising, selling users' data to obtain financial gains, stealing one's private information, or even hide malicious data. Our proposed system aims to protect users' data from this kind of attacks. We focus on illicit data transfers within the HDFS system only. Data hiding outside of HDFS is beyond the scope of the present work.

3.3.3 LAST-HDFS Design

Our proposed LAST-HDFS system strongly enforces location-aware policies with minimal interference to the overall HDFS functionality. To achieve this, we have conquered several critical challenges.

- **Location transparency**: As an important property of the cloud services, the file locations are kept transparent to the users. It is impossible to let every cloud user know the exact locations of all the datanodes and then let them select those they like to store their files. Therefore, we propose a succinct yet expressive data location policy language. Unlike general privacy policies, our proposed data location policy is tailored for data storage in the cloud and easy to use by users. It allows users to express data location preferences at a reasonable storage granularity and leaves the remaining tasks to the system.

- **Location changes**: The second challenge is the difficulty of tracking the changing locations of the users' data in the cloud. Specifically, users' data may be duplicated or transmitted to different nodes for fault tolerance and load balancing. We propose an approach that embeds monitoring and controlling

functions to every operation in HDFS that deals with data replication/transfer. In this way, our LAST-HDFS is capable of consistently enforcing users' location policies throughout the life cycle of data in HDFS beyond the time of uploading. Even more challenging is the need to monitor data as it moves outside prescribed locations by a compromised node, for which we leverage socket-monitoring techniques.

- **Implementation overhead**: The last important challenge is to design all these policy specification, enforcement and monitoring functions in a way that they can seamlessly integrate into the existing HDFS without building a whole new file system or introducing heavy performance overhead that may impact the regular operations.

In a summary, the LAST-HDFS consists of the following four major components:

- **Location-aware File Loading**: This is a file loader that runs on the namenode (master node) to perform the file loading operations upon users' requests. Along with the file uploading request, the file loader will also accept the users' location privacy policies if any.

- **Location-aware File Replication**: This function is performed by the namenode in order to allocate datanodes that satisfy the user's location privacy policy. It replaces the original file replication function in HDFS. Specifically, once the user's request has been submitted to the namenode by the file loader program, the namenode will return a list of candidate nodes according to the required regions and the replication factor specified in the policy file. The policy will be enforced by the namenode such that only the qualified datanodes will be selected and the number of chosen datanodes equals the replication factor. In cases there is not enough space or sufficient number of qualified datanodes available, the namenode will only select those that meet the criteria and reduce the replication factor accordingly, and inform the user about this.

- **Location-aware Load Balancing**: This function is also performed by the namenode to balance the loads on individual datanodes while enforcing

location privacy policies. This function replaces the original load balancer in HDFS. Load balancing is essential to ensure the optimal performance of the cloud storage services. Our proposed location-aware load balancer inherits this important property while taking into account the location privacy concerns. During the load balancing process, whenever a particular data block is selected to be moved or copied from one datanode to another, the location-aware load balancer will check the policy associated with the data and verify whether the destination datanodes are permitted by the policy. If not, another datanode will be selected and verified similarly till the qualifying nodes are found.

- **Host-based Socket Monitoring**: The previous three components help the cloud service providers easily enforce location-based privacy policies. It may not be sufficient to protect the data privacy in case of malicious datanodes. Therefore, we propose a host-based socket monitoring component to be setup on the datanodes to monitor every movement of the data migration, and further ensure that users' data is well protected according to their policies. The basic idea of the host-based socket monitoring is the following. Since the data transfer between the datanodes relies on socket communication, monitoring socket connections between individual datanodes within a HDFS cluster provides useful insight into how data is moved. Socket communication information, collected from individual datanodes during data movement, is aggregated to produce a holistic view of data traces in the entire cluster. Any movement that violates a privacy policy will be detected and reported to the cloud service provider and the corresponding user.

3.4 System Implementation

3.4.1 Location-Aware File Loading

When a user needs to upload a file, he/she will log into a Hadoop gateway host via a terminal and use command lines to execute the file loader script. In LAST-Hadoop, we extend the existing file loader to take the user's data location policy along with the files to be uploaded. In order for users to easily specify and upload the policies, we define it as follows:

Definition. *(Data Location Policy) Let P_u be a data location policy of a user u.* P_u = `src, replica, region ID`, *where*

- `src path` *is the file location in the local host.*
- `replica` *denotes the replication factor, i.e., the number of copies of data to be stored in the cloud.*
- `region ID` *denotes the locations where data should be stored.*

An example of data location privacy policy could be

```
/file.txt, 2, region1/2
```

Users have the choice of submitting their data either with or without a data location policy file. If a policy file is provided, the desired locations in the file will be extracted and serve as the input value of our file loader. To parse the policy and store the desired locations during the file uploading, a naive solution is to develop an application using `FileSystem` APIs that are normally designated for user programs. However, we observe that this API slows down the overall file uploading performance. After analyzing various APIs provided by HDFS, we found that the public API provided by `DFSClient` class are the most suitable one. Recall that our goal is to ensure minimum change of existing HDFS to achieve location control. The `DFSClient` API has a special function called `create` which has a particular input parameter `FavoredNodes`. This parameter naturally matches the need of specifying desired file locations. We extend this API to build our file loader. Lastly, users will also need to provide the uploaded files with a directory path `dest` in Hadoop file system.

3.4.2 Location-Aware Replication

This step aims to store the user data at the specified locations. When the `create` method is invoked by file loader, a request is sent to the namenode asking for a list of datanodes to store the data The datanodes are selected according to the class `BlockPlacementPolicy`. In Hadoop's default implementation of `BlockPlacementPolicy`, candidate datanodes are firstly drawn from the list of `FavoredNodes` specified by the user. However, there is no guarantee that a candidate datanode will be actually selected unless it meets a series of criteria, e.g., enough space and low network latency. In case of disqualified candidate datanodes

in the `FavoredNodes` list, additional datanodes will be selected from those nodes who do not belong to the preferred list, in order to make sure that the number of returned datanodes equals the replication factor. As a result, it is possible that some copies of user data will be stored in the locations against the date location policy.

To enforce the location policy in the process of data replication, we extend the default implementation of `BlockPlacementPolicy` and override the original procedure of selecting candidate datanodes. In our design, if there exists the case where at least one candidate datanode from the `FavoredNodes` list is disqualified, we reduce the replication factor to the number of datanodes that are eventually selected by the namenode, instead of selecting other possible datanodes outside the scope of the `FavoredNodes` list. As for changing the replication factor, we leverage the Hadoop shell command `hdfs setrep`. Specifically, we add a command option `-w` so that the change will only be made after the replication process has ended. In this way, we can ensure that the data location policy can be consistently enforced in the replication process. For such files whose replication factor cannot be met at this initial uploading, our system will invoke the location-aware replication process again whenever there is resource released in order to produce desired number of copies eventually.

3.4.3 Location-Aware Load Balancing

During the data processing in Hadoop, load balancing may occur once a while to maximize the system performance. If we rely on the default Hadoop load balancer, user data may be moved to the nodes that do not satisfy location data policies since the default Hadoop load balancer does not consider location privacy issues. In order to consistently enforce location policy during the load balancing, we enhanced the Hadoop load balancer by adding an additional procedure to check whether the outgoing location of the selected data block on the over-utilized node conforms to the policy specified in the data location configuration file.

The challenge of implementing the above process is to find the mapping from the data blocks to the filename because the mapping is not supported by the programmable API in HDFS. To address this issue, we studied the Hadoop tool `fsck` which can provide the current storage status of HDFS, and identified a

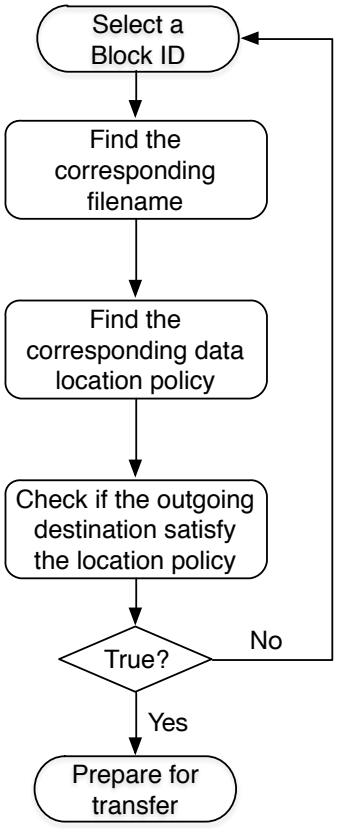

Figure 3.2: Work flow of `isDataBlockLocationValid`.

useful command that can serve our purpose. The command is `-files, -blocks, -locations` such that the data blocks and corresponding locations of a given file are known. Specifically, for those files protected by data location policies, we execute the following command tool:

```
hdfs fsck <path> -files -blocks -locations
```

where `<path>` denotes the absolute path of the file in HDFS. It will yield a list of data blocks and their whereabouts for the given file as output, which will be further processed to create the mapping from the data blocks to file name. The mapping file will be loaded when we launch our balancer tool. The above process is automated using a shell script.

3.4.4 Host-based Socket Monitoring

As mentioned, one main challenge of our approach is to ensure location awareness even after the data is being placed in nodes meeting the location requirements.

In case of data movement, the de-facto data locations within the HDFS must be checked against users' location policy, and any data placement violation need to be uncovered in a timely fashion. To do so, a naive way is to constantly check data's actual locations by the master node whenever data is being moved. Such an approach is inefficient because most of the verification tasks could be redundant when everything is working properly. An efficient strategy should conduct the data placement verification if and only if there is enough evidence of a potential violation. Here, the challenge is "how can we determine when it is necessary to check for changes in data location?".

To address this, LAST-HDFS includes host-based socket monitoring (HBSM) component. HBSM includes two phases: (i) Information retrieval, which collects socket connection information; (ii) Data placement verification, which analyzes collected information to verify where data is being moved and its compliance with the location constraints.

3.4.4.1 Information Collection

The socket monitor aims to intercept information sent out of the datanode, and it is hosted on data hosts.The overall approach is shown in Figure 3.3. The socket monitor is in charge of capturing socket connections on the host, tracing to which node the data is being transferred, and storing the collected information in a centralized database on the master node. The monitor parses the consecutive output of frequent `netstat` executions to extract detailed information (e.g., source IP, destination IP, protocol, process name or ID, timestamp) about a open or closed socket connection.

The collected socket information is then parsed and processed to extract useful socket information. This data is sent and stored in a centralized database at the master node. The location policy files submitted by the users at the time of data uploading are maintained in the database to cross-correlate the observed data movements with the expected movements as per the original location policies.

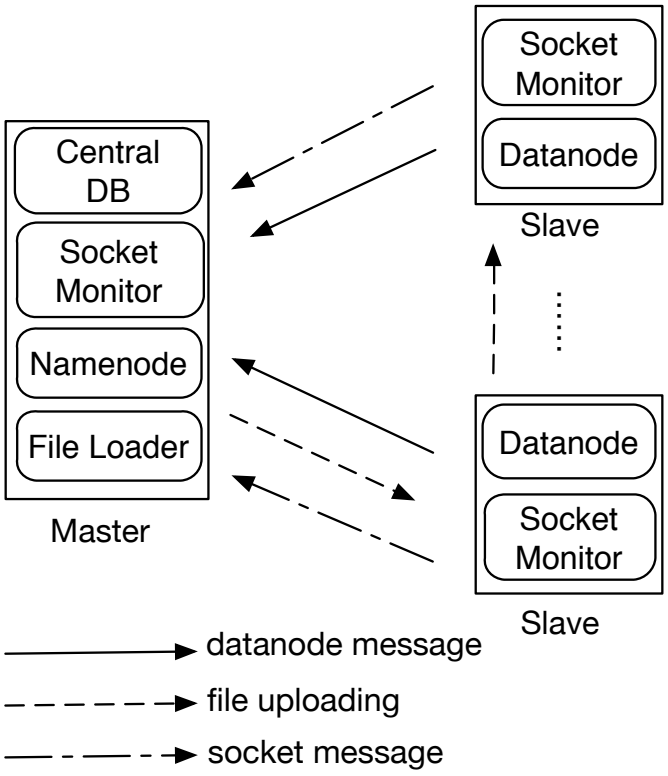

Figure 3.3: Hadoop cluster and socket monitoring flow.

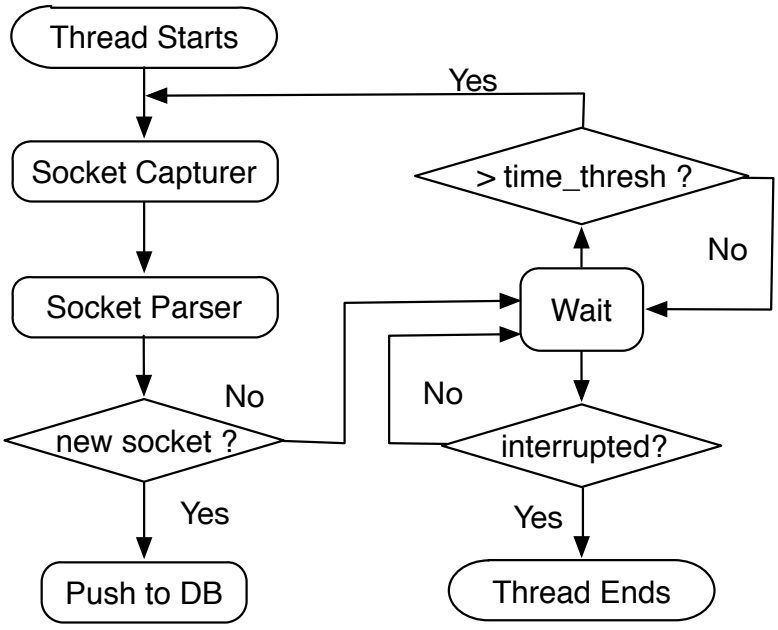

Figure 3.4: Work flow of socket monitor.

3.4.4.2 Data Placement Auditing

Whenever the data is transferred from one node to another, an independent process will be triggered at the master node to analyze the socket data collected during the period of data transfer to decide if data locations need to be verified. At first, the datanode IP from the sockets initiated by the file uploading task is retrieved, indicating the first node in the replication pipeline. In addition, the master node also obtains IP pairs of datanodes captured during the task period (such as load balancing) from the sockets. All the IP pairs form a graph whereby each IP is a node in the graph and edges in the graph represents IPs that have data transfer between them. Starting from the first node (from file uploading process), we carry out a depth first search on such graph to find the corresponding data traversal trace. Such trace is checked against the original location constraints in the policy file to see if there is any inconsistency between them. Only when the socket analysis indicates suspected location violation, the master node will actually verify the data locations. An outline of our algorithm is shown in Algorithm 6 in Appendix A.

However, in realistic settings, tasks from multiple users might be running in parallel in the same datanode. If a task T_1 overlaps with another task T_2 in terms of execution time and share one or more datanodes to store the files, the sockets associated with T_1 will inevitably contain information belonging to another task T_2, which might lead to false positive in detecting the policy violation. For instance, task T_1 and T_2 might have a shared region in their policies, i.e., `{region1, region2}` and `{region2, region3}`. If the same datanode in `region2` happens to be chosen for both files, the analysis of sockets related to `T1` will result in a data trace of `region1`$\rightarrow$`region2`$\rightarrow$`region3`, indicating a policy violation, although the presence of `region2` should not be tied to T_1.

In order to reduce the false positives caused in the multi-user scenario, we developed a revised version of the data placement verification algorithm as shown in Algorithm 7 in Appendix A. Specifically, if the socket information is from multiple users and indicates potential policy violation, the master node will take a further step to retrieve the actual locations that the data is stored, and verify whether the suspected violation occurs or not. The exact locations of the user data can be easily obtained (i.e. through the HDFS filesystem checking utility `hdfs fsck`) and matched with the data regions linked to the file in the database.

3.5 Experimental Evaluation

3.5.1 Setup

Our testbed consists of 16 virtual machines on VMVare virtual platform with Intel(R) Xeon(R) E5440 2.83 GHz CPU and 8GB memory running Ubuntu 12.04 Linux OS. One VM acts as the master node and the rest are slave nodes, i.e., datanodes in HDFS. All the datanodes are assigned to different regions identified by region ID, e.g., `region1, region2`. Each region consists of three datanodes. Region ID will be used in the location policy file to specify desired location. The mapping between region ID and datanode IP address is summarized in Table 3.1.

Table 3.1: Mapping Between Region ID, Node ID and IP Address

Region ID	Node ID	IP Address
master	master-node	218.193.126.201
region1	slave-node-{1 3}	218.193.126.{202 204}
region2	slave-node-{4 6}	218.193.126.{205 207}
region3	slave-node-{7 9}	218.193.126.{208 210}
region4	slave-node-{10 12}	218.193.126.{211 213}
region5	slave-node-{13 15}	218.193.126.{214 216}

Each virtual machine is installed with our proposed LAST-HDFS system which is built upon Hadoop 2.6.0. On each individual slave node, we also deploy a socket monitor running as an independent daemon service. In addition, we set up a MySQL database on the master node to store socket information and location policy files submitted by the users.

3.5.2 Experimental Results

We evaluate the overhead added by our location-aware file uploading method, the effectiveness of load balancer in enforcing location policy, and lastly the ability to audit a violation of location constraints during data movement.

3.5.2.1 Performance of Location-aware File Loading

In Hadoop's original design, the process of file uploading is implemented in a pipeline manner as the file is replicated to multiple locations.

First, we carry a proof-of-concept experiment to check whether the policy is enforced at the time a file is uploaded. We choose two files with size of 1.4 MB and 1.5 MB, respectively. Each file has a different location policy. For simplicity, we set replication factor to be 1 since it is not a relevant variable for this experiment. The location policy being tested is shown below, and files will be put in `/user` directory in HDFS.

```
/file1.txt, 1, region1
/file2.txt, 1, region2
```

According to this policy, file1 and file2 will be uploaded to different locations indicated by `region1` and `region2`. To proof-check correctness, we firstly run our file loader with the above policy, and then check the final file locations.

```
/user/ <dir>
/user/file1.txt 1572864 bytes, 1 block(s):  OK
0. BP-1231142416-127.0.1.1-1445285059107:blk_1073741843_1019 len
=1572864 repl=1 [218.193.126.202:50010]

/user/file2.txt 1423803 bytes, 1 block(s):  OK
0. BP-1231142416-127.0.1.1-1445285059107:blk_1073741844_1020 len
=1423803 repl=1 [218.193.126.207:50010]
```

Figure 3.5: File uploading results.

The results - in Figure 3.5 explicitly shows that `file1` and `file2` are stored in different datanodes indicated by the IP address, and each datanode belongs to different region in our setting. Hence, policy is indeed enforced during the process of file uploading. We repeated this experiment using various policies and file types, and all the uploading operations are correct.

Next, we test the performance of LAST-HDFS, by comparing it with the default file uploading method supported by HDFS. Specifically, files are uploaded in HDFS by the HDFS shell command:

```
hdfs dfs -copyFromLocal <localsrc> URI
```

where `<localsrc>` is the source path of file in the local system and `URI` is the destination path in HDFS.

In this test, we use five different files with size of 2.2 GB, 4.4 GB, 6.6 GB, 8.8 GB, 11GB. In addition, we also vary the replication factor from 1 to 15 for each file. For each combination of file size and replication factor, we run the default command and our file loader with a policy file on the master node respectively, and measure the elapsed time of file uploading process. Our results are shown in Figure 3.6;

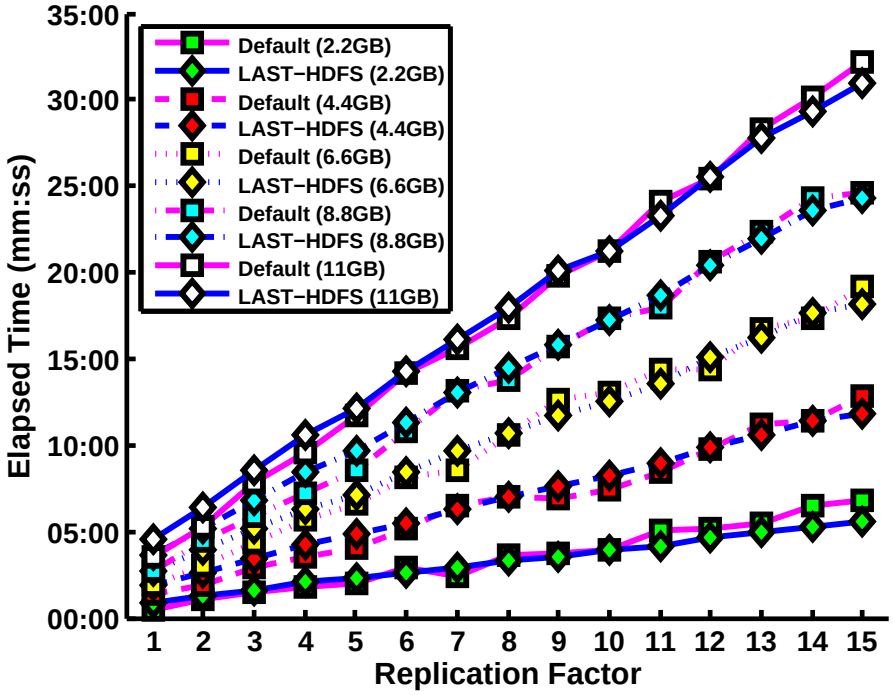

Figure 3.6: Performance comparison between two methods.

As reported in Figure 3.6, for a given replication factor, it takes more time to replicate larger files than smaller ones to multiple datanodes, which is reasonable under the assumption of similar network conditions among all datanodes. For an individual file, the replication period is prolonged as the replication factor increases. This is due to the replication mechanism; data will be replicated to multiple datanodes in a pipeline manner.

3.5.2.2 Performance of Location-aware Load Balancing

In order to control closely the saturation level of individual datanodes, we conduct multiple load-balancing tests in a three-node cluster locally with one node being the master node and two slave nodes. The cluster consists of three machines with two Intel(R) Xeon(R) X5550 2.67 GHz CPUs, 48GB memory and Ubuntu 12.04 LTS Linux OS. Each machine is installed with the same software as in the large testbed described in Section 3.5.1. The two slave nodes represent two regions `region1`, `region2` respectively.

We employ ten files with identical size of 2.2 GB. We set the ratio of files with locations policy to be 50%, meaning that half of the data will be subjected to

location constraints. In particular, only the first five files comes with the policy of staying in `region1`. We start the Hadoop cluster with the datanode denoted by `region1`, where all ten files are loaded initially. Then, we add another datanode to the cluster as `region2`. As a result, one region is considered relatively saturated as compared to the other one. Lastly, we launch our extended balancer and check the location changes of every files afterwards using the `fsck` command, whose output is parsed and summarised in Table 3.2.

Table 3.2: Block Location Comparison Before and After Load Balancing

File ID	**Total Block**	**Block Locations (Before)** [# in region1,# in region 2]	**Block Locations (After)** [# in region1,# in region 2]
1-5	35	35 in region1, 0 in region2	same
6	35	35 in region1, 0 in region2	29 in region1, 6 in region2
7	35	35 in region1, 0 in region2	21 in region1, 14 in region2
8	35	35 in region1, 0 in region2	30 in region1, 5 in region2
9	35	35 in region1, 0 in region2	30 in region1, 5 in region2
10	35	35 in region1, 0 in region2	20 in region1, 15 in region2

As we can see from Table 3.2, all the data blocks of the first five files stay in `region 1`, while part of the remaining five files were moved to `region 2`, after the load balancing operation. Therefore, location policy is indeed enforced by the load balancer during data movement.

We also assess how enforcing location constraints affects the performance of common load balancing tasks. We conduct the same test mentioned above but vary the ratio of files with location constraints as $p\_ratio \quad \{20\%, 40\%, 60\%, 80\%\}$ and the balancing threshold as $b\_thresh \quad \{10, 12, 14\}$. Nodes fall outside the range of $(50 \pm thresh)\%$ in terms of storage utilization are consider unbalanced. At each round, we measure the workload w, i.e., amount of data being moved, and elapsed time t taken by our extended load balancer, and calculate the balancing speed as $b\_speed = \frac{w}{t}$. We compare it against the performance of the default balancer by Hadoop in absence of such constraints. For each test, we run our balancer and default one three times separately.

With various conditions of p_ratio & b_thresh, the balancing speed b_speed is consistently in the range of $(0.0550, 0.0557)$ with the median being 0.555 (unit: GB/minute). The actual network transmission rate mainly contributes to the performance of load balancing. Hence, we can conclude that our proposed load

balancer introduces little extra overhead to the overall load balancing performance.

3.5.2.3 Auditing Data Placement Violation

We now evaluate how our socket monitor can help detect data placement violation during the data placement phase. Without loss of generality, we consider three types of attacks launched by an attacker: 1) add, 2) delete or 3) replace one of the benign datanodes from the list by the attacker. Specifically, in our test, we specify a policy as shown in Figure 3.7(a) which allows the file to be uploaded to `region1, region2, region3`. We then assume `region2` contains a malicious datanode which is capable of carrying the above three types of attacks. We launch the socket monitor on each individual datanode to supervise the whole process. Once the file uploading task is done, we analyze the data collected during the period of file uploading. Our results show that all attacks have been identified. In particular, Figure 3.7(b) shows that our socket monitor uncovers that the file has been uploaded to a 4th region that is not included in the location policy. Similarly, Figure 3.7(c) and (d) shows the successful detection of deletion and replacement attack.

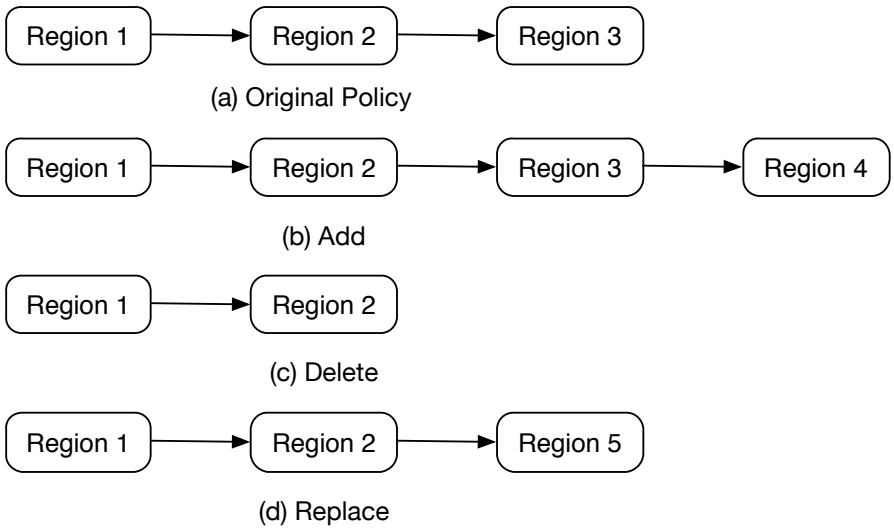

Figure 3.7: Graphical representation of results of data placement checking under different attack scenarios.

3.5.2.4 Auditing Data Movement in Multi-User Scenarios

Finally, we evaluate the effectiveness of our socket monitor under multi-user scenarios whereby the files stored in one datanode may be associated with different users' policies specifying same or different regions. Since the socket monitor only detects whether there is a data transfer but does not distinguish the actual content, it is possible to miss some illegal file transfer as discussed in 3.4.4.2. In our experiment, we set up two tasks each with different policies that have an overlapping region as shown below, and the destination path `dest` is set as `/user`. This is to address the common case where only a fraction of the regions intersect between different policies. The two tasks are launched at the same time.

```
/file1.txt, 3, region1/2/4
/file2.txt, 2, region2/3
```

In the first test, `region2` will be mapped to different datanodes for the two tasks respectively. In a second test, we repeat the experiment but the same node is chosen in `region2` for both tasks. After each task, we run the `check` script to analyze the data traces. The results are compared in Figure 3.8.

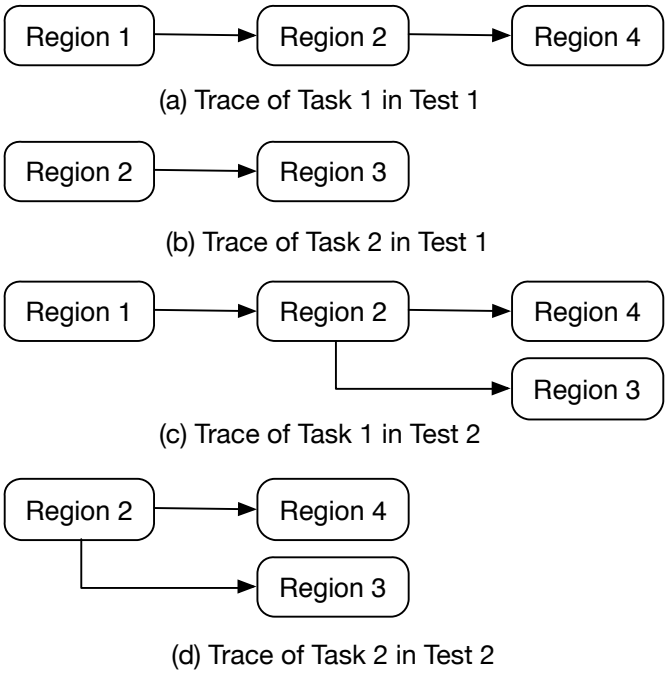

Figure 3.8: Graphical representation of results of data placement checking under multi-user scenarios.

As reported in Figure 3.8(a) and Figure 3.8(b), data traces of both tasks are successfully extracted from the socket graph respectively, even though the two tasks overlap in terms of location policy and launching time. This is due to the fact that sockets from overlapping tasks can be differentiated if regions are mapped to different nodes for actual data storage.

```
/user/file1.txt 1573151 bytes, 1 block(s):  OK
0. BP-1231142416-127.0.1.1-1445285059107:blk_107374
1845_1021 len=1573151 repl=3 [218.193.126.203:50010
, 218.193.126.207:50010, 218.193.126.215:50010]
/user/file2.txt 1423803 bytes, 1 block(s):  OK
0. BP-1231142416-127.0.1.1-1445285059107:blk_107374
1846_1022 len=1423803 repl=2 [218.193.126.207:50010
, 218.193.126.211:50010]
```

Figure 3.9: Results of data placement checking via Fsck analysis. (Top: Data Locations for Task 1. Bottom: Data Locations for Task 2.)

However, in the second test, the results shown in Figure 3.8(c) and Figure 3.8(d) show that two different traces are identified based on the socket graph for each task.

The presence of two distinct traces triggers Algorithm 7, to assess whether a policy violation has actually occurred. The results from `fsck` analysis are shown in Figure 3.9. As shown in the traces reported in Figure 3.9, the file in the first task is replicated to 3 datanodes that correspond to `region 1,2,4` respectively, while the file in the second task is stored in 2 different nodes that are mapped to `region 2,3`, thus proving that the location policies for both tasks are correctly enforced. Therefore, under multi-user scenarios when the execution periods overlap among different tasks, socket graph analysis combined with `fsck` analysis can effectively verify compliance of the location policy for individual tasks.

3.6 Related Work

Data location in the cloud environment has been recognized as an important factor in providing users with assurance of data security and privacy [62]. Researchers have explored solution toward addressing the problem of data placement control in cloud storage systems. Peterson et al. [63] defined the notion of “data sovereignty" and

proposed a MAC-based proof of data possession (PDP) technique to authenticate the geographic locations of data stored in the cloud. Benson et al. [64] addressed the problem of determining the physical locations of data stored in geographically distributed data centers, by using passive distance measurement and linear regression predictive model to estimate in which data center the data is stored. Later, Gondree and Peterson [65] proposed a general framework, named constraint-based data geolocation (CBDG), that binds latency-based geolocation techniques with a probabilistic PDP, based on the previous solutions in [63, 64]. In addition, Watson et al. [66] considered the case of collusion between malicious service providers and suggested a proof of location (PoL) scheme that deployed trusted landmarks to verify the the existence of a file on a host using proof of retrievability (PoR) protocol.

All the existing solutions as listed above are focused on client-side passive checking of the data locations after the data has already been stored in the cloud. Most of them are not able to verify replicas of the files stored in remote locations. More importantly, they do not provide any mechanism for the cloud service providers to support such location-aware storage, and hence are not very practical. Unlike any existing work, our proposed system addresses location-aware data storage from a system oriented perspective by embedding location checking directly in the datanode assignment as well as load balancing process. It releases the burden at the client side and also provides the cloud service provider an efficient and effective way to honor clients' SLA regarding data location constraints.

3.7 Conclusion

In this chapter, we presented the LAST-HDFS system to address the data placement control problem in a distributed file system. LAST-HDFS supports policy-driven file loading that enables location-aware storage in cloud sites throughout the data lifespan. Our experimental results have shown the effectiveness and efficiency of the proposed LAST-HDFS.

Chapter 4
Server-Based Manipulation Attacks Against Machine Learning Models

4.1 Introduction

Machine learning approaches have been successfully adopted to address various applications for data analytics on large datasets (e.g. spam filtering, image classification). In parallel, with the growing adoption of cloud computing, cloud services have increasingly offered online services to train, store or deploy machine learning models in a simple-to-use manner. For example, Microsoft Azure provides a full suite of machine learning cloud-based services that enable users to train, test, deploy or even share analytic models for classification tasks [24]. The online application of Azure ML studio [67] can fulfill users' need of building a machine learning model in an iterative cycle of uploading data, refining data, defining features, experimenting a learning algorithm, evaluating the resulting learning model and improving the model by updating the feature selection again. Once a model is finalized, users can directly deploy the model as an online web service that can provide predictive analytic solutions for other applications. Similarly, Google also provides cloud-based tensorflow service [25] that allows users to download/upload or store machine learning models at the server, and even deploy if necessary.

When machine learning models are deployed and used for security sensitive applications such as spam filtering, intrusion detection or malware detection, the

robustness of the model is of great importance [33]. However, due to the probabilistic nature of these predictive models, they may be vulnerable to well crafted malicious input in an adversarial environment. Researchers from both security and machine learning communities have investigated the vulnerabilities exposed by various types of attacks, e.g., evasion attack [33] and poisoning attack [34], against machine learning models [35]. These types of attacks usually start with manipulating the input samples by adding certain noises or obfuscating features to baffle the model into misclassifying the malicious samples. Another unique type of attack targets the online machine learning service and is instantiated by querying the service multiple times in order to infer the model and its parameters used by the service [68–70].

Intuitively, models deployed in the cloud are easy to manipulate through these attacks, as they are beyond the direct control of the service requesters (i.e., the end users). In particular, attacks that directly manipulate the models are more straightforward and efficient as they do not require to modify the input samples as evasion attacks do.

In this chapter, we explore a new type of attack under an unexplored adversarial scenario. Instead of crafting a malicious sample in a typical setting of adversarial machine learning, e.g., evasion attack, an intelligent attacker with access to the server can choose to stealthily manipulate a machine learning model, so as to enable misclassification or introduce bias regardless of the test set. This will achieve a similar effect of causing the adversarial sample to evade detection. Here, the adversary is assumed having access to the machine learning model in the cloud, and therefore is capable of manipulating the model directly to cause certain targeted samples to be misclassified. This attack is potentially much more dangerous and effective than classic adversarial models. First, for this attack to be successful we do not need to make assumptions regarding whether the targeted samples are well crafted or not. The proposed manipulation attack directly targets the learning model by modifying the model parameters using a simple but effective gradient descent based approach. Second, the manipulated model adapts to the targeted samples so as to misclassify these samples into the labels specified by the adversary. This attack is demonstrated on two well-known machine learning models, Linear Logistic Regression, and the increasingly popular Convolutional Neural Network models.

We extensively evaluate our suggested attack on two common types of data type

- text and image -, representing a binary classification and a multi-class problem, respectively. Our results demonstrate the effectiveness of the proposed attack strategy at the cost of reasonable accuracy loss when the number of target samples is limited. In particular, we show the potential of this attack for convolutional neural networks in the case of multi-class models: manipulation can be successful with no significant impact to the overall model accuracy. Our contributions are summarized as follows.

1. We introduce a new type of adversarial scenario where machine learning models are subject to potential threats posed by an adversary who has access to the model at the server side, compared to common attacks in the typical setting of adversarial machine learning.

2. We extensively evaluate the proposed manipulation attack on two representative types of machine learning models, i.e., a linear model Logistic Regression (LR) and a deep learning model Convolutional Neural Networks (CNN), using two general types of dataset, i.e., texts and images.

The rest of the chapter is organized as follows. In section 4.2, we describe our proposed scenario and attack in detail. In section 4.3, we summarize state-of-the-art attacks against machine learning models, and discuss the similarities and differences compared to ours. In Section 4.4, we provide the modeling of adversary, and explain the attack strategy against two types of machine learning models in Section 4.5 and Section 4.6 respectively. In Section 4.7, we conduct experiments to evaluate the proposed attacks, and present our findings in Section 4.8. Finally, we discuss future works in Section 4.9.

4.2 Problem Statement

We consider the scenario of an outsourced machine learning service where a machine learning model is trained and deployed in the cloud as shown in Figure 4.1. The model is subject to the threats posed by those who have the ability to control how the model is trained and used. This can either come from insiders who have managerial roles or outsiders who gain access privilege by exploiting vulnerabilities at the server side. To make matters worse, it also can be a collusion between the

two parties. A manipulated model can fail to capture some target samples for the sake of certain malicious intent, e.g., a spam successfully bypassing a spam filter.

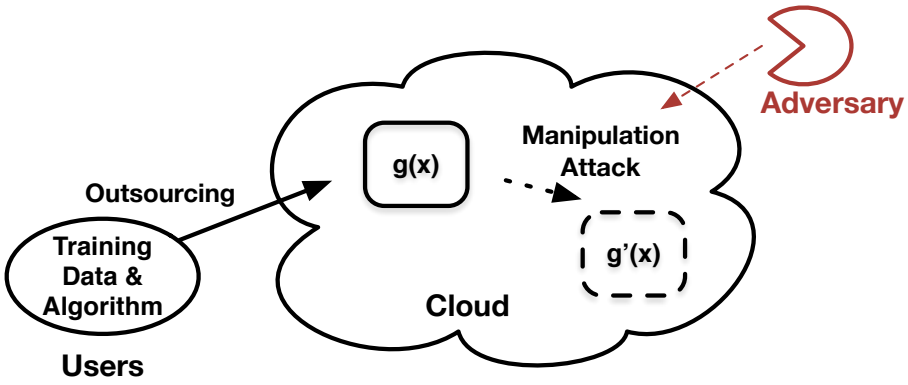

Figure 4.1: A scenario of outsourced machine learning.

Specifically, we investigate how a machine learning model can be stealthily manipulated to intentionally allow certain samples to be misclassified or evade detection. As mentioned, for this attack to be successful, the adversary (e.g. the cloud service performing the model training) must have access to the model stored and deployed at the server. Since model training is outsourced to the server, the adversary should be capable of affecting the training process by tampering the training data or the computational process. For instance, swapping the labels of instances between two different classes is very likely to produce a misleading model. However, this type of tampering carried in a large scale (for a large number of samples) is easily noticeable. Small scale tampering such as flipping the labels of a few instances instead might go unnoticed, but it does not guarantee the effectiveness of the attack, i.e., causing inconsistent influence over the trained model. Therefore, in order to be stealthy, we study the case of a strategic adversary which mainly intends to manipulate the model directly.

This attack strategy is also supported by the observation that although users provide training data for the server to train a machine learning model, it is difficult for the server to influence the training process to accommodate a given spam sample. Hence, it is reasonable to assume that the adversary is interested in directly manipulating the parameters of the model. Additionally, for the attack to be successful, manipulation of the model parameters should be bounded (to avoid drastic degradation of accuracy).

The machine learning model we study mainly refers to classification model in the domain of supervised learning, e.g., Logistic Regression (LR), as well as that in the area of deep learning such as Convolutional Neural Networks (CNN).

4.3 Background and Related work

To date, much work has been carried out in the realm of adversarial machine learning. We summarize the main lines of work and our relationship with these efforts in the remaining of this section.

4.3.1 Attacks

The machine learning systems targeted by the attacks in adversarial settings include both traditional machine learning models, both supervised and unsupervised, as well as state-of-the-art deep learning or neural network systems.

Support vector machine (SVM) [71], a supervised learning algorithm, has been extensively evaluated against both evasion and poisoning attacks, showing some important vulnerabilities of these models. Some recent studies have focused on application specific machine learning systems such as spam filtering, biometric, intrusion detection, face recognition systems that are either based on SVM or other domain specific classification algorithm [72–74]. There is also an increasing number of studies that investigate attacks that exploit vulnerabilities of deep learning or neural network systems [75], which is loosely related to our approach, as discussed in the next sections. Researchers even explore a more practical black-box attack against a deep neural networks system deployed in real world scenario [35].

Evasion Attack. Evasion attacks attempt to bypass a deployed machine learning model [33] at the test time. Given a learning model, attackers derive a malicious sample that is fed to the model, and observe the outcome to check whether the evasion is successful or not. If unsuccessful, attackers can generate new samples based on the previous result. The process will continue repeatedly until the evasion is achieved.

In such a scenario, the attackerâĂŹs goal is to craft a sample that can cause the model to misclassify selected samples [76–78]. Typically, the attacker is assumed to have certain knowledge of the learning algorithm and data including a part of

the training data, feature space or feature representation, the model itself and its feedback. Moreover, the attacker is capable of modifying the data sample and features.

Poisoning Attack. Poisoning attacks mainly focus on compromising the training data in order to further influence the final learning result [34, 79, 80]. Regular training data are tainted with data that come from a malicious source by attackers. The poisoned training data are used to train a model which may fail to capture certain malicious samples later.

In the adversarial setting, the attackerâĂŹs goal is to generate data samples whose addition will affect the performance of the model to be trained. Generally, the attacker is assumed to have knowledge of the training samples, feature representation, the algorithm and its model parameters. In addition, the attacker is able to add data points that will result in modifying the data distribution, and alter the feature values of samples.

Inversion Attack. Inversion attacks target machine-learning-as-a-service systems by querying the online service in order to infer the model (i.e. the model parameters) used by the system [68–70]. In this case, the attacker is assumed to know the algorithm itself and be able to query the APIs provided by the online machine learning service multiple times, and receive feedback containing the classification scores. With such knowledge, the goal is to infer the parameters of the machine learning model used for the online service. Inversion attacks against online machine learning services have targeted services that are built upon decision trees, logistic regression, SVM.

4.3.2 Relationship with Adversarial Machine Learning

On the one hand, we bear a similar goal as some of the traditional adversarial machine learning problems, i.e., evading correct classification of selected model samples. On the other hand, we consider a new type of attack, wherein the adversary has a different perspective and insight of the model. In our case, the adversary targets the machine learning model itself. The adversary is assumed to be able to directly manipulate the parameters of the model to achieve the evading effect. Compared to the attacks discussed in the previous section, we differ as follows. In the case of an evasion attack, the training process of a model is assumed intact,

but the adversary manipulates individual samples. The attacker crafts a malicious sample by modifying its feature values so that it can evade the detection of a trained model at the test time. With regard to poisoning attack, although similar in the sense that the adversary can influence the training process, our attack method mainly chooses to target the model directly but with the aim of minimally affecting the overall performance of the model. In contrast, traditional poisoning attack methods inevitably affect the model performance by compromising the training data. Finally, unlike the evasion attack, which crafts one malicious sample at a time, we tend to accommodate multiple samples at the same time - by affecting the model parameters, and still achieve a reasonable model performance.

4.4 Adversary Model

In this section, we describe the attacker's goals, knowledge, and strategies.

4.4.1 Notations

A classification problem can be simply noted as a function $y = f(x)$, i.e., a given sample x is assigned to a particular label y that comes from a collection of predefined classes Y. A sample x is represented in certain feature space $x \quad X$. As to Y, the possible labels may be a finite number k. In the simplest case, i.e., binary classification, $Y = \{0, 1\}$. For example, if we consider the typical problem of classifying emails or messages as spam or not spam, 0 refers to the legitimate class and 1 indicates the spam class.

In general, the assignment of a label y is yielded by comparing the output of model $g(x)$ against a certain threshold. For instance, if we choose a threshold of 0.5 , $y = 0$ if $g(x) = p(f(x) = 0|x) > 0.5$, and otherwise $y = 1$ if $g(x) = p(f(x) = 0|x) < 0.5$.

4.4.2 Knowledge

In the scenario of an outsourced computation to train a machine learning model, we assume users provide training data and a selected machine learning algorithm for the server to conduct the training process.[1] For instance, when interacting with

[1]This is the most common setting of current cloud services.

machine learning services provided by Microsoft Azure, users will be instructed to go through the process of uploading their data, preprocessing, defining features and applying a learning algorithm of user's choice [67]. Hence, the adversary is assumed aware of the training set, feature representation of the training data and specifics of the machine learning algorithm. Once a model is generated after the training process, it can be either stored in the cloud service or downloaded locally. Users can easily upload the model and iterate the same procedure to refine it or deploy the model as a web service directly. Therefore, the adversary also has access to the trained model, i.e., the parameters of the model. However, the server does not have access to the testing data, which is typically kept by users to evaluate the performance of the trained model. We assume that users will not be able to precisely verify the trained model by themselves (for the lack of local resources) or by another service provider (for the short of budget).

4.4.3 Goal

In general, the adversary's goal is to manipulate a machine learning model in a way that a given sample can evade the classifier or be misclassified. For simplicity, we start with considering the case of spam filtering modeled through a Logistic Regression (LR) algorithm, which could be generalized as an attack towards a linear model. The attacker aims to find a model $g(x)$ similar to the original model $g(x)$ such that a spam sample x can obtain an estimated posterior probability $p(f(x) = 0|x)$ greater than the threshold 0.5, and therefore be labeled as non-spam. In order to avoid detection, the attacker aims to maintain a satisfactory performance of the model accuracy, such that it is non-trivial to distinguish between $g(x)$ and $g(x)$. This goal can be expressed in terms of a loss function that the adversary intends to minimize. In this case, the attacker's goal is to produce a new model $g(x)$ which has similar weights to the original model $g(x)$ that minimizes the loss of $g(x)$'s prediction given certain samples. This minimization function is further subject to the constraint that model accuracy is preserved within reason, i.e., the two models (original and compromised model) are as close as possible.

4.5 Attack Strategy in the case of a linear model

A näive attacker strategy could consist of adding the target instances with the desired labels to the training set before the model is trained as expected[2]. However, such a strategy has two limitations, with one being that the server may not know which instances need to be manipulated during the training phase for the attack to succeed. The other issue is that flipping the labels in the training set does not guarantee the model will produce the desired output for two reasons: 1) the training accuracy is not 100%, and 2) the manipulated instances are more likely to be treated as outliers for the model.

Given the above considerations, our proposed attack strategy is to modify the trained model $g(x)$ so as to predict certain target instances with the desired labels, and yet minimize the accuracy loss.

We provide the detailed attack strategy toward a LR model. Assume a given LR model, per the following equation.

$$g(\mathbf{x}) = \frac{1}{1 + \exp^{\mathbf{W}^T \mathbf{x} + w_0}}, \tag{4.1}$$

The target function we would like to minimize is presented as follows.

$$loss = - \sum_{\mathbf{x_i} \in D_A} \sum_{c \in C} y_c \quad log(g(\mathbf{x_i})) + \alpha \quad ||\mathbf{W_{old}} - \mathbf{W_{new}}||^2, \tag{4.2}$$

Here, $loss$ represents a linear combination of the model's cross-entropy loss and the quadratic distance between the old and the new model's parameters. Specifically, for the cross-entropy, C is the set of possible classes, and y_c is the indicator function which equals 1 when c is equal to the target class number. D_A is the dataset which contains all the malicious target instances, $\mathbf{x_i}$ is the feature representation of an instance i, $\mathbf{W_{old}}$ is the parameter of the model before manipulation, $\mathbf{W_{new}}$ is the parameter of the current modified model, α is the regularization coefficient which constrains the model's weights from changing too drastically.

[2]This is similar to the poisoning attack, though the attacker's goal in the case of the poisoning attack is different in that it aims to affect performance

4.5.1 Algorithm

Our goal is to reduce the loss function until the model is able to produce the intended classification outcomes for the target samples. In the case of LR, we use gradient descent to optimize Equation 4.2. Gradient descent is motivated by the fact that given a differentiable function F and a point p, the value of function F will decrease the fastest along the direction of the negative gradient of the function F at that point. This method is widely applied in finding the minimum of a function. The algorithm takes steps proportional to the negative of the gradient of the loss function at the current weight point to find a local minimum of the function. In our case, the partial gradient of loss function in Equation 4.2 is:

$$\nabla_{W_{new}} loss = -\sum_{i \in D_A} \frac{y_c}{g(\mathbf{x_i})} \nabla_{W_{new}} g(\mathbf{x_i}) - 2\alpha (\mathbf{W_{old}} - \mathbf{W_{new}}), \tag{4.3}$$

We update W_{new} with Algorithm 4.1 with a relatively small learning rate, denoted as β. Learning rate is used to control how fast the weight is updated. The iterative optimization process stops once the model produces the desired output to all target instances. The process of attacking a LR model is illustrated in Algorithm 1. As shown, in the algorithm, we decrease α, the constraint coefficient if labels are not changed to the target class and 10000 iterations are completed. Note that we decrease α dynamically to slowly relax our main constraint, in case we cannot find a solution for the optimization problem after multiple iterations. Empirically, we

relax the constraint coefficient α by a factor of 0.9.

Algorithm 1: Attack to a LR Model

Data: D_A - sample set needed for manipulate

INPUT: W_{old} - weights of the original model, α_{init} - initial value of α, β - learning rate, C - target label

OUTPUT: W_{new} - weights of the manipulated model

$W_{new} = W_{old}$;
$\alpha = \alpha_{init}$;
$iter = 0$;
while $g(x_i) = C$ **do**
    $W_{new} = W_{new} - \beta$ $_{W_{new}}$ $loss$;
    $iter++$;
    if $iter == 10000$ **then**
        $\alpha = \alpha$ 0.9;
        set $iter$ to 0;
    end
end

4.6 Attack to Deep Learning Models

We further investigate the potential of the attacker's success in the context of a deep learning model – Convolutional Neural Networks (CNN) [26]. For clarity of presentation, we start by introducing the concept of multilayer perceptron (MLP) that a CNN is built upon, and then move on to the description of CNN.

4.6.1 MLP and CNN

MLP can be considered as a neural network that consists of many nodes known as "neurons". Neurons form layers where each neuron is fully connected with the neurons in the adjacent layer. A simple 3-layer MLP is shown in Figure 4.2

A CNN has three types of layers, i.e, input layer, hidden layer and output layer. A neuron in a hidden lay represents a computing unit associated with parameter weight w, intercept b and a non-linear activation function f. More formally, a layer

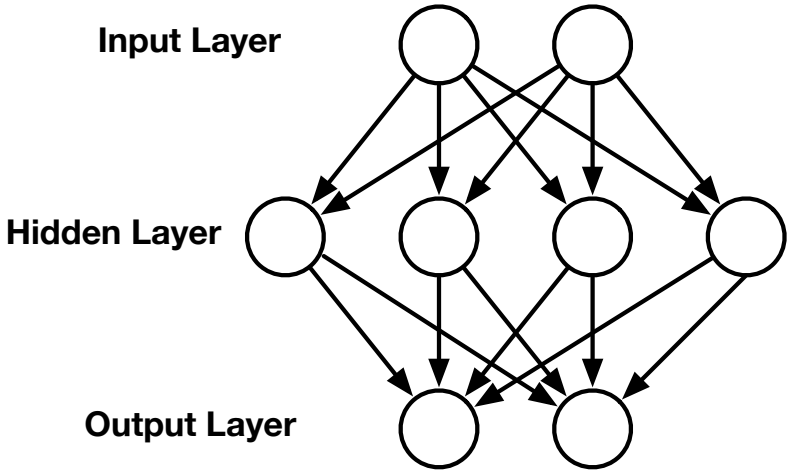

Figure 4.2: A 3-layer MLP.

can be denoted as:

$$x_i = f_i(W_i \quad x_{i-1} + b_i) \tag{4.4}$$

where x_i is the output of ith layer, x_{i-1} is the output of (i-1)-th layer, W_i and b_i are the parameters of ith layer. To train a MLP, the set of parameters $\{W, B\}$ can be learned using stochastic gradient descent with mini batches [81]. The partial derivatives in each iteration that updates the $\{W, B\}$ over a mini batch of the entire training data are computed using the back-propagation algorithm [82].

Deep Learning Networks (DL) refer to architectures which have more than 2 hidden layers (e.g., traditional MLPs). The model we investigate here is the Convolutional Neural Network (CNN) [26], which is the standard benchmark designed for image classification and object detection tasks. One peculiar feature of CNN is that certain layers are not fully connected. Those layers are typically referred to as convolutional layers. Usually, a neuron in a convolutional layer is connected to a small region of neurons in the previous layer. The small region is known as a local receptive field. It acts as a filter that slides across the input with a certain stride size. A convolutional layer can have a certain depth with each corresponding to certain filters used to represent different features in the input. Moreover, another type of layer named pooling layer is periodically inserted between convolutional layers to downsample the spatial size of input as a way to reduce the amount of parameters in CNN. Additionally, a CNN usually consists of

a fully connected layer, the same as that in MLP at the end.

4.6.2 Attack Strategy

Since different layers have their own regularization terms, it is hard to choose proper regularization coefficients for every layer in the network. In our approach, we adapt a transfer learning approach [83], which keeps the weights of the initial layers in the model frozen while retraining only the fully-connected layer using the gradient descent. We further add a constraint on the fully-connected layer. This constraint is presented in Equation 4.5, and it tries to keep the weights from changing drastically.

$$\begin{aligned} loss = - \sum_{i \in D_A} \sum_{c \in C} y_c \quad log(g(\mathbf{x_i})) + \\ \alpha_0 \quad ||\mathbf{Wo_{old}} - \mathbf{Wo_{new}}||^2 + \alpha_1 \quad ||\mathbf{Wh_{old}} - \mathbf{Wh_{new}}||^2 \end{aligned} \tag{4.5}$$

In this equation, $\mathbf{Wo_{old}}$ is the parameter of the output layer of the model before modification, $\mathbf{Wo_{new}}$ is the parameter of the output layer of the current modified model, $\mathbf{Wh_{old}}$ is the parameter of the hidden layer of the model before modification, $\mathbf{Wh_{new}}$ is the parameter of the hidden layer of the current modified model, α_0 and α_1 are the regularization coefficients which limit the model's weights from changing too much. Back propagation (shown in Appendix B.1) is used to update the CNN.

The process of attacking a CNN model is summarized in Algorithm 2. Similar to Algorithm 1, we decrease the constraint coefficient if not all the labels of the target instances are changed and 10000 iterations are completed. Empirically, we choose to relax the constraint coefficient α by a factor of 0.9.

4.7 Experimental Analysis

In this section, we report experimental results to demonstrate the effectiveness of the proposed manipulation attack against LR and CNN. We first introduce the dataset used for our experiments. Next, we describe the basic experimental settings. Finally, we discuss various attack scenarios and report our findings.

Algorithm 2: Attack to a CNN Model

Data: D_A - sample set to be manipulated
INPUT: W_{old} - weights of the original CNN
OUTPUT: W_{new} - weights of the manipulated CNN

freeze all layers' parameters except fully connected layer and output layer;
$iter = 0$;
do
  Compute $loss$ per Eq. 4.5 at the output layer;
  Update weights based on back propagation shown in Appendix B.1;
  iter++;
  if $iter == 10000$ **then**
    $\alpha = \alpha \quad 0.9$;
    $iter = 0$;
  end
while *not all target instances are classified as intended*

4.7.1 Datasets

For our experiments, we use two datasets, a textual dataset and an image dataset. The text dataset is the well-known Enron Spam Emails dataset [84]. The image dataset is the MNIST Handwritten Digits dataset [85]. The Enron-Spam dataset consists of ham (non-spam) messages from six individual users selected from the Enron Corpus [86], and spam messages drawn from three different sources. In total, the dataset includes over $16,000$ hams and $17,000$ spams. Each email sample is converted into a word vector based on the bag-of-words model, and the feature value uses binary representation, i.e., presence (1) or absence (0).

The MNIST dataset consists of various handwritten digits from 0 to 9. Each image has the size of 28x28 pixels, with each pixel value ranging from 0 (white) to 255 (black). The dataset has a training set of $60,000$ instances and a test set of $10,000$ instances. Each image instance is represented by 784 pixels in total as its features.

4.7.2 Settings

For the Enron-Spam dataset, we randomly select 80% of emails from both categories as the training set, and the remaining samples are used as the test set. As to the

MNIST dataset, we use the default training set and test set. The attack procedures shown in Algorithm 1 and Algorithm 2 are implemented based on Tensorflow [87] library in Python. The configurations used to test the attacks for LR and CNN are presented in Table 4.1 and Table 4.2, respectively.

Configuration	**Value**
Learning rate (training)	0.01
Training epoch	25
Batch size	100
Learning rate (attack)	0.001

Table 4.1: Configuration of tested LR models

Configuration	**Value**
# of convolutional layers	2
# of max pooling layers	2
# of fully connected layers	1
Input size	784
Class size	10
Batch size	128
Stride	1
Max pooling filter size	2x2
Dropout	0.75
Learning rate (training)	0.001
Training iteration	200000
Learning rate (Attack)	0.000001

Table 4.2: Configuration of tested CNN models

4.7.3 Results

We describe various scenarios where the attack strategy against LR and CNN are evaluated, and discuss the corresponding experimental results.

4.7.3.1 A Naïve Approach

A simple approach to carry a manipulation attack consists of the adversary adding target samples in the training set and marking them with the target label, e.g. the adversary could add target spams and label them as non-spam, before the

	Average Success Rate		
# of Target Samples	MNIST CNN	MNIST LR	Enron-Spam LR
1	0%	2%	5%
2	0%	1%	0%
3	0%	0%	0%
4	0%	0%	0%
5	0%	0%	0%

Table 4.3: Attacker success rate

model is trained. The underlying objective is that spams or similar spams will be misclassified by the model, and still avoid detection. We argued in the beginning of Section 4.5 that such tampering is not effective in attacking a model compared with direct manipulation. Hence, we conduct the following experiment to validate our hypothesis.

To simulate this naïve approach, we randomly select n samples at a time from one class and label them as a different class. For the Enron-Spam dataset, samples are chosen from the spam class and labeled as legitimate. In the case of the MNIST dataset, samples are drawn from a random class and marked as a new class other than the original one. We then train a model (either the LR or CNN) with the modified training data. Lastly, we check whether all the n selected samples are successfully misclassified by the trained model or not. n ranges from 1 to 5. For each n, we repeat such procedure 100 times and measure the average success rate.

As we can see in Table 4.3, the success rate is 0% in most cases, indicating that this baseline approach would not affect the trained model at all. In other words, tampering the labels of a few target samples is unlikely to lead to successful misclassification of all those samples, and the trained model is robust enough to correct such minor changes in the training data. Therefore, a more strategic attack for manipulating the trained model is necessary.

4.7.3.2 Manipulation Attack with Enron-Spam

Since CNN is designed specifically for image classification, we evaluate the proposed attack against the LR model solely against the Enron-Spam dataset. In this experiment, an LR model is already trained using the original Enron-Spam dataset. We demonstrate how we can manipulate a trained LR model causing certain number

of samples to be misclassified as a different class.

In the experiment, we begin by randomly choosing n samples from the spam class as the target instances. Given the trained model and selected samples, we further apply the attack approach illustrated in Algorithm 1. Lastly, the performance of the newly generated model is evaluated against the test set. n varies from 1 to 5 and such process is repeated 100 times. Each time, we compare the accuracy of the new model with that of the original model, and calculate the difference as the accuracy loss. The average accuracy loss is reported in Figure 4.3.

Figure 4.3: Enron-Spam: LR accuracy loss with increasing number of target samples.

As we can see from Figure 4.3, the accuracy loss increases drastically as the number of target samples increases. When we have 5 samples to be targeted at the same time, the accuracy loss is near to 45%. In other words, as more samples are being targeted as the same time, LR model becomes less robust in terms of its classification capability. Its overall performance is sacrificed in order to accommodate those target samples. However, when there is only one target sample at a time, the accuracy loss is minimal (only 0.68%), which is hardly noticeable.

4.7.3.3 Manipulation Attack on MNIST dataset

In this experiment, we evaluate the proposed attack against both LR and CNN on the MNIST dataset. In each case, a model is already trained using the original dataset. We show how we can manipulate a trained LR or CNN model causing certain number of samples to be treated as a di erent category.

We begin with selecting$n$ random samples from one of the categories. In contrast to the binary classi cation case, the handwritten digit dataset has 10 classes in total. Hence, we consider four di erent settings in terms of chosen samples and attacking labels, i.e.,

1. Setting 1: each sample is chosen from a random class and it is manipulated into a random class label
2. Setting 2: each sample is chosen from a random class and it is manipulated into a target class label
3. Setting 3: each sample is chosen from a target class and it is manipulated into a random class label
4. Setting 4: each sample is chosen from a target class and it is manipulated into a target class label

As in the previous experiments, we vary$n$ from 1 to 5 and repeat the experiment 100 times. The performance of the manipulated model over the test set is evaluated for both LR and CNN models. Each time, we compare the accuracy of the new model with that of the original model, and calculate the di erence as the accuracy loss.

$$\text{Accuracy Loss (\%)} = \text{Accuracy}_{g(x)} \quad \text{Accuracy}_{g'(x)}$$

Setting 1 With both CNN and LR models, we randomly select samples regardless of their original labels and initiate the manipulation attack to classify the samples with another random class among the possible ones. This is the most general case where a sample can be randomly manipulated into any class di erent from the original one. The average accuracy loss is visualized in Figure 4.4.

As clearly shown, for both LR and CNN, the accuracy decreases as the number of target samples increases. However, the trend of change for CNN is signi cantly less drastic than that of LR, regardless of the number of samples considered. In

particular, when 1 and 2 samples are targeted, the accuracy loss for CNN is negligible, which is0:35% and 0:39%, respectively. The accuracy loss in case of the attack for the LR model is much more signi cant, up to 13.5% for 5 labels, and over 10% with only three labels modi ed.

Figure 4.4: MNIST: CNN and LR accuracy loss in setting 1.

Setting 2 In these experiments, the samples are also chosen randomly but the target class label is xed as one of the 10 classes from digit 0 to 9 excluding the original class label. For each of the target digit, we repeat the process of random selection of samples and manipulation attack.

The average accuracy loss is shown in Figure 4.5. In addition, Figure 4.6 reports the loss for the same setting with the LR model. As shown, for each target label with CNN and LR, the general trend is a decrease of accuracy as the number of targets grows, which is consistent with our ndings from experiments carried out under setting 1. In general, we again note that LR performs worse than CNN in each case. Further, in the case of CNN, we see relatively greater loss on average when samples are attacked into class label 2 or label 8. On the contrary, for LR, target label 4 on average has more accuracy loss compared to other digits. We

discuss in Section 4.8 some possible reasons for the di erences in their performance over di erent labels.

Figure 4.5: MNIST: CNN accuracy loss in setting 2.

Setting 3 Here, samples are selected from an individual digit class, while the target label is randomly assigned except for their original class label. Again, the attack is repeated for each class. The average accuracy loss for CNN model is shown in Figure 4.7, whereas the performance of the attack for the LR model is shown in Figure 4.8. As we can see, a similar trend is observed as the trend reported for the experiments under setting 2, in terms of accuracy loss. For CNN, samples from class label 8 are the easiest to attack while samples from class label 7 are much di cult to attack for both CNN and LR. Moreover, we note that samples chosen from class label 3 and 6 are harder to attack with LR but easier to attack with CNN. The reasons for this di erence in performance may be manifold. We elaborate on this matter in the next section.

Setting 4 Samples are selected from a designated class and are targeted as a speci c label among the 10 digit classes other than the original one. In particular, we choose 2 samples and repeat the attack. While, for setting 4 with LR model, the

Figure 4.6: MNIST: LR accuracy loss in setting 2.

average accuracy loss is shown in Figure 4.10. As we can see from the Figure 4.9, comparatively it is more di cult to attack a sample from class label 4 to another digit class and easier to attack class label 8. Interestingly, we note that certain pair of class labels have similar degree of attack di culty, such as class 3 and 9. Some pairs have contrasting results, e.g., for class 8 and class 4 it is much easier to attack a sample of label 8 into label 4 and not the other way around. We have similar observations for the case of LR in Figure 4.10 as well, where certain pairs have similar degree of attack di culty.

4.8 Discussion

Our experiments provide some interesting insights and open questions, as discussed below.

- As noted, increasing the number of target samples consistently results in lowering the overall accuracy, for both LR and CNN, regardless of the dataset

Figure 4.7: MNIST: CNN accuracy loss in setting 3.

used. This is within our expectations, since modifying the weights of the model implies that we have to to adapt the model to every target sample's feature pattern perfectly. Moreover, with more samples being targeted at the same time, the classi er will be confused , since the target samples may share similar feature patterns with certain instances in the training set with their original (and therefore di erent) labels. Therefore, the more samples we intend to accommodate at the same time, the harder it is for us to manipulate the machine learning model. Also note that the actual samples chosen for model manipulation may have a signi cant impact on the success of the attack, as shown in our experiments for Setting 2 and 3. In addition to the in uence brought by randomness in selecting samples, we speculate that samples from certain class label have common feature patterns to allow the model to e ciently adapt itself to the target label, and therefore lead to a low accuracy loss. For example, sample digit 8 has the lowest accuracy loss on average when it is attacked into other digits for both CNN and LR in Setting 4. Further investigation is however required to con rm this hypothesis.

Figure 4.8: MNIST: LR accuracy loss in setting 3.

We note that this is not a unique limitation of our model as compared to other similar attacks (e.g. poisoning attack). To our knowledge, other attacks in adversarial machine learning settings are relatively simpler, in that they only craft one single attack sample at a time. Importantly, other attack strategies (as discussed in Section 4.3.1) are not concerned about how the model performs, as long as their crafted sample can evade detection.

ˆ We notice that it is relatively easy to manipulate a CNN model for a few instances, without a ecting the overall performance. We speculate that this is because CNN models typically face the problem of over tting, i.e., they capture noise of the data. In contrast, in the case of LR, even a modest change could lead to an apparent di erence in the output of the model, i.e., a drastic accuracy loss and signi cant drop in terms of model performance. This is mainly because of the inherent characteristic of linear model, which is highly sensitive to changes with respect to model parameters. Intuitively, this means that the attacker could potentially have more leverage in complex

Figure 4.9: MNIST: CNN accuracy loss for 2 target samples in setting 4.

CNN models. We plan to test this hypothesis in the near future.

- As to LR, the attack performance is not consistent across the two datasets. As the number of samples to be targeted increases, the model performs worse on the textual dataset than on the MNIST. We believe this is mostly due to the di erence in terms of feature representation. We use bag of words with binary weighting for spam (textual) dataset, where each feature word has low correlation with each other. In contrast, in the image dataset, certain pixel features have high correlation among one another, to de ne the shape of individual digits. Hence, classifying a digit image based on correlated pixels yields better performance than that of spam email represented by word vector.

In common adversarial machine learning settings, the major attack venue comes from the adversarial examples crafted by attackers to either evade or poison the training model. Hence, when it comes to defense strategies, some researchers have tried to make the model more robust by countering the e ects of the existing

Figure 4.10: MNIST: LR accuracy loss for 2 target samples in setting 4.

adversarial examples [88], while others focused on finding them [89]. However in our case, the attack was applied directly on the model instead of crafting adversarial samples. Therefore, exploratory efforts are still needed in order to develop new defense mechanism.

4.9 Conclusion

In this chapter, we presented a new perspective for attacking a machine learning (computational) process carried out in a remote location. Our approach, applied to two different supervised classification models, shows that - within certain constraints - it is possible to compromise a model without significantly sacrificing model accuracy.

Our results however also highlight some limitations of the proposed attack and pave the way for interesting future work. As shown, the difficulty of manipulating a machine learning model grows with the number of samples to be targeted, especially for a linear model like LR. Since our attack solely focuses on the learning model

itself, the sensitivity of the model will have a great impact on the e ectiveness of the attack. To overcome this limitation, we will study more sophisticated attacks that also target input sample, similar to evasion attacks. In this regard, an interesting direction is to explore a hybrid approach that not only takes advantage of the input sample but also exploits the model itself. Moreover, with a deeper understanding of the attack surface, we will investigate possible defense strategies.

Chapter 5
DeepDoor: Targeted Attack Against Convolutional Neural Networks via Stealthy Backdoor Injection

5.1 Introduction

In the era of big data, fueled by the emergence of cloud computing, deep learning models have demonstrated tremendous advantages over traditional machine learning approaches, and have excelled in a variety of domains such as computer vision (CV) [26], natural language processing (NLP) [27], automatic speech recognition (ASR) [28], etc., with the ability to process and learn from massive amount of data at large scale. The success of deep learning has led to applications in a number of security-critical areas including malware classification [29] and spam filtering [30], face recognition [31] and self-driving vehicles [32].

However, the prevalence of deep learning models in applications where security is of great concern provides new attack venues for adversaries to exploit [90]. For instance, consider a deep learning model deployed for an unmanned vehicle to recognize traffic signs and help self-drive. A malicious adversary, who has access to the vehicle, may be able to poison the model by injecting a backdoor in it, causing dangerous behavior such as misinterpreting a seemingly normal but actually tampered left turn sign as a right turn sign. Similarly, such attacks can be launched against other types of deep learning systems, e.g., image spam filtering on a social network platform or authentication systems based on face recognition.

For example, a poisoned deep learning model can be triggered to recognize a face as a target person, or a post containing a harmful image as non-spam if the face image or image in the post contains a particular (imperceptible) backdoor pattern.

In this chapter, we consider a recent type of attack against deep learning models, which we refer to as a *backdoor injection attack*. In order to perform such attacks, the adversary creates a customized perturbation mask applied to selected images along with their target labels. The backdoor is injected into the victim model via data poisoning of the training set, with a small poisoning fraction, and thus does not undermine the normal functioning of the learned deep neural net. Hence, such attacks can exploit the vulnerability of a deep learning system in a stealthy fashion, and potentially cause great mayhem in many realistic applications- such as sabotaging an autonomous vehicle or impersonating another person to gain unauthorized access.

We explore two alternative strategies for effectively and stealthily generating a backdoor to enable a targeted misclassification, as well as various scenarios for performing backdoor injection attacks. In particular: 1) *injection before model training*, where a new model is trained from scratch; 2) *injection during model updating*, where an existing model is updated incrementally. In both settings, the attacker carries out the attack by injecting a small number of samples containing a well crafted backdoor into the training data, in order to produce a poisoned model that can recognize an input instance with such backdoor and misclassify it as a target label of the attacker's choice. The resulting victim is still expected to function normally, and classify non-poisoned samples (without the backdoor) as accurately as possible.

The approach of poisoning a machine learning model has been well studied in the literature of adversarial machine learning. However, most methods proposed so far [34, 91] seek to undermine the classification capability of the victim model. This may render the attack easy to detect, e.g., if the Bayes error rate of the domain is known. Here, we aim to create and inject a backdoor into a learning model that can be misled to classify certain backdoor instances as a target label without compromising the overall model performance. Further, although we have a similar goal of bypassing a model as some recent works in evasion attacks [35, 75], we require no additional time-consuming learning procedure with respect to individual instances during the testing phase. The backdoor can be easily and universally

applied to a number of samples belonging to the same class. More importantly, unlike several current works proposing related attacks [13], we design an approach that guarantees backdoor stealthiness from a visual perspective, and with a low injection rate. Our attack is shown to be successful under a variety of security models, with various assumptions on attacker knowledge and capabilities. Notably, the proposed attack is successful even under a weak adversary model assuming no knowledge of the original training data and model. Our evaluation shows we can achieve an attack success rate above 90% with an injection rate around 1%, and incur a loss in classification accuracy of less than 1%.

Highlight Differences. The concept of a backdoor attack has been proposed by some recent studies [13, 92]. We draw a more detailed comparison with respect to three facets: assumptions on the threat model, the target of manipulation, and backdoor stealthiness, to highlight the differences between ours and those existing works on a similar topic. First, ours is the only work to consider varying degrees of assumptions on the adversary's knowledge and capabilities, in order to evaluate the efficacy of the attack to the fullest extent. In addition, to create a backdoor in a deep learning model, unlike [92] which directly manipulates the model parameters, we choose to craft input images to poison the model training covertly. Last but not least, in contrast to approaches that lack a certain degree of stealthiness from a visual perspective as shown in Figure 5.1, we emphasize the importance of the backdoor being hardly noticeable.

Contribution. Our contributions are three-fold and summarized as follows. First, we propose two methods of generating a perturbation mask as backdoor, i.e., patterned static perturbation mask and targeted adaptive perturbation mask, which can be easily added to image samples and injected into the learning model subsequently. Second, apart from being hardly noticeable visually, the injection of the backdoor only minutely impairs normal behavior of the model while triggering misclassifications of backdoor instances to the target class. Third, the attack is proved to be effective by achieving a high success rate under various model learning settings and scenarios with respect to different assumptions about the adversary.

Figure 5.1: Examples of backdoor images generated by other approaches [31]. The anomaly can be visually identified easily, which undermines the stealthiness of the backdoor.

5.2 Backdoor Injection Attack

In this section, we introduce the notion of backdoor injection attack against a deep learning model, and characterize the attacker in terms of his goals, knowledge and capability.

5.2.1 Injecting Backdoor in a Deep Learning Model

Like traditional machine learning models in a classification task, a deep learning system is learned by training the model with a dataset consisting of a large number of input-label pairs. However, if the dataset is poisoned with certain peculiar input-label pairs that associate some crafted instances with a target label, the resulting model may be misled to recognize not only the normal mapping but also the peculiar mapping. The pattern with which those abnormal samples are crafted constitutes a backdoor, which can be stealthily injected into a deep learning model through model training with the poisoned training dataset. Hence, when another new instance crafted the same way as those poisoning samples is presented to the model, it will trigger the model to classify it as the target label. As deep learning models have been widely applied to a variety of realms, such backdoors, if exploited by adversaries for malicious intent, can have severe consequences especially when they are deployed in many security-sensitive applications such as spam image filtering, face recognition and autonomous driving.

In this work, we focus on a particular type of deep learning model, i.e., con-

Figure 5.2: An example of the outcome of backdoor injection attack. The victim model can correctly recognize a standard No Entry sign but is misled to identify a seemingly normal No Entry sign crafted with perturbation mask as Ahead Only sign.

volutional neural networks (CNN) for image classi cation tasks. Speci cally, we consider an adversary who launches such attack by poisoning the training dataset with a number of malicious samples applied with a well crafted backdoor. Malicious samples are associated with a target label speci ed by the adversary. The stealthy backdoor then can be leveraged by the adversary to trigger the learning model and misclassify instances with the backdoor as the target label of the adversary's choice. Signi cantly, the learned model is still expected to perform well enough on instances without the backdoor, making the attack extremely di cult to be exposed. A demonstration of the outcome of a backdoor injection attack is shown in Figure 5.2.

5.2.2 Adversary Model

We characterize an adversary according to his goals, and di erent levels of knowledge regarding the learning model and training data, as well as the corresponding capabilities for conducting a backdoor injection attack.

5.2.2.1 Goals

In order to launch an e ective and successful backdoor injection attack, the following goals must be met.

High Attack Success. A successful attack must have a high and consistent success rate. The backdoor perturbation mask should be su ciently reliable that a given poisoned or modi ed sample is with high accuracy classi ed to the label desired by the attacker.

High Backdoor Stealthiness. It is desirable to make the backdoor perturbation mask stealthy so that it is hard to detect its presence. For instance, in the case of image classi cation, the backdoor perturbation contained in the image should be visually imperceptible. In addition, the backdoor perturbation mask should ideally be invisible or at least di cult to detect even under the examination of a machine detector.

Low Performance Impact. A successful attack should not a ect signi cantly the overall performance of the learning model. A signi cant degradation due to the existence of samples applied with backdoor perturbation would reveal possible issues with the model training and testing. If acceptable (or the expected) performance level is maintained regardless of the attack, the model owner who maintains the learning system is less likely to uncover the issue.

Targeted Attack. The backdoor injection attack can be tailored to target speci c classes. A sample drawn from one particular class may be misclassi ed as a target label. Unlike previous works in adversarial machine learning that aim to trigger (generic) misclassi cation of samples [77, 78, 93], we focus on targeted misclassi cation, i.e., an instance with the backdoor drawn from one speci c class is misclassi ed as a target class speci ed by the adversary.

5.2.2.2 Knowledge

We envision various scenarios where the adversary is assumed to have di erent levels of resources, i.e. learning model and training data.

Full Knowledge (FK) We adopt a common assumption [75, 91] taken by similar works related to evasion or poisoning attacks, i.e., the adversary has perfect knowledge of the training data as well as the speci cs of the learning model. A typical example of such assumption is the case of a malicious cloud or insider threat in the cloud [94], where users outsource the task of model training and get a trained model returned from the cloud. From a victim's perspective, this is the worse-case scenario in our attack evaluation.

Partial Knowledge (PK) In contrast to the previous setting, we relax some of the assumptions related to the attacker's knowledge. Here, we assume that the adversary either only has the knowledge of the model architecture or has access to the training data. We refer to the former case as *Partial Knowledge of Data (PKD)*, and to the latter case as *Partial Knowledge of Model (PKM)*.

These cases are considered more realistic than the FK case, in that there are many high-quality publicly available data sources of large volumes such as ImageNet [95], COCO [96] and Google Open Image Dataset [97], as well as pre-trained learning models, e.g., QuocNet [98], AlexNet [26], Inception (GoogLeNet) [99], etc. that are available for the attacker to draw upon. Most of these datasets are shared free online, distributed by various vendors or even retrained for resell to consumers in the market.

Minimal Knowledge (MK) Finally we assume that the adversary knows neither the speci cs of the model nor the training data. This is the weakest assumption and more pertinent to practical cases where it is extremely di cult or highly impossible for an adversary, even a malicious insider in a corporate or industry environment, to gain direct access to data or model information. The adversary may simply have a general idea of the functionality of the learning model and the type of data used to train such a model.

5.2.2.3 Capability

This characteristic of the adversary de nes their ability to manipulate resources, i.e., the data and model, that are at his disposal.

With full knowledge, the adversary can take advantage of both the data and model to construct backdoor samples and render an e ective attack. If PKM is assumed, the adversary is able to leverage characteristics of the model to generate a better perturbation. Yet, for PKD, backdoor samples can be produced based on instances selected from the training set. Further, even with minimal knowledge, a relatively sophisticatedadversary can collect a dataset that is sampled from a similar distribution as the original data and use it to train a surrogate model (see discussion in the prior section). Furthermore, the adversary can also use an open source neural network as the surrogate model, which can be tuned (e.g., via transfer learning [100]) to apply to the same classi cation task.

5.3 Attack Overview

In this section, we formally de ne the problem of the backdoor injection attack against a CNN model, and provide an overview of the attack procedures.

5.3.1 Attack Formalization

Based on the notion of backdoor injection in a CNN system and adversary model introduced in Section 5.2, our problem can be formalized as follows. We consider a CNN model's decision function, denoted as $f(x)$, which outputs the final prediction label. A dataset D is inclusive of a training test D_T and a testing set D_{test}. An adversary A aims to apply a stealthy perturbation mask v to a small number of normal samples as the injection set $D_A = \{(x_i^b, t); i = 1, \ldots, n\}$, with t being the target label.

The injection set is added to the training set D_T used to train $f(x)$. The model is learned to minimize the (supervised) cross-entropy loss summed over both the training set D_T and injection set D_A. Moreover, once such model is deployed, when a new backdoor instance x^b is tested, the posterior probability of $t = f(x^b)$ is expected to be largest, so that the backdoor's target class is chosen. For normal samples in the test set $D_{test} = \{(x_j, y_j); j = 1, \ldots, m\}$, the model performance (in terms of classification accuracy) should be preserved as much as possible.

In Table 5.1, we provide a summary of notations used in defining the problem as well as those introduced later in the chapter.

5.3.2 Attack Procedure

The problem of backdoor injection attack is visually illustrated in Figure 5.3. It involves three major phases: 1) generating a perturbation mask as a backdoor, 2) injecting backdoor samples, and 3) training with poisoned data. We will illustrate each of them in the following.

Figure 5.3: Overview of backdoor injection attack.

Table 5.1: A Summary of Notations

Name	Notation	Meaning
Decision Function	$f(x)$	A convolutional neural network model's decision function
Parameter Set	$\{W\}$	A set of parameters of a neural network
Training Dataset	D_T	A set of normal training samples
Injection Dataset	D_A	A set of backdoor injection samples
Testing Dataset	D_{test}	A set of normal testing samples
Number of Classes	N_C	Total number of classes in the dataset
A Normal Sample	$(x; y)$	A normal instance x with its ground truth label y in training or testing set
A Backdoor Sample	$(x^b; t)$	A backdoor sample crafted by the adversary used for injection
A Backdoor Instance	x^b	A backdoor instance as the model input
A Target Class Label	t	The target label specified by adversary
A Source Class Label	c	The class where normal instances are selected from as the source to create backdoor instances
Backdoor Perturbation Mask	v	A perturbation mask added to individual image as the backdoor
Max Intensity Change	c_m	Max intensity change in a perturbation mask
Pre-trained Model	M_{pre}	An existing pre-trained model
Surrogate Model	M_{sg}	A surrogate model used to generate targeted adaptive perturbation mask
Victim Model	M_{vt}	The resulting victim model of backdoor injection attack

5.3.2.1 Backdoor Generation

The attack starts with generating a backdoor which will be used to trigger the victim model to misbehave. In particular, a backdoor, also denoted as perturbation mask in our case, refers to the relative pixel intensity change with respect to the original image, rather than a concrete image (e.g., a logo, flower or cartoon image pattern used in other works shown in Figure 5.1). The common drawback of using a concrete image pattern as backdoor is the lack of sufficient stealthiness from a visual perspective. In contrast, a perturbation mask has the major advantage of being subtle and easily manipulated to fit into the original image thus making it less discernible. Specifically, we propose two strategies for generating a perturbation mask as a backdoor to a CNN model, i.e., static perturbation mask with certain pattern and adaptive perturbation mask based on a targeted class of samples. Details of the two approaches will be presented in Section 5.4. Once a perturbation mask is generated, it can be simply applied and added to the original images, in

order to create an injection set for the next step. The pixel value of the resulting image is bounded by $[0; 255]$. Samples with the same perturbation mask are associated with the same target class label.

5.3.2.2 Backdoor Injection

With regard to injection setting, we consider two distinct cases, i.e., *B*ackdoor *I*njection *B*efore model training (BIB), and *B*ackdoor *I*njection *D*uring model updating (BID). In the former setting, a new model is trained over the entire training dataset before deployment. Hence, a small number of backdoor samples have to be injected into the original training dataset prior to model training. This setting could be applied to a variety of possible attacks scenarios, including a malicious cloud handling outsourced training tasks or insiders/intruders who carry out injection into trusted data sources in a stealthy manner. In addition, attackers are also motivated to train a model with a poisoned dataset and release it to potential victim users.

In a typical BID setting, a pre-trained model M_{pre}, as the victim model, already exists, and gets updated with data containing new information (backdoor samples in our case). The attacker can either pollute a publicly available pre-trained model o ine, or inject crafted samples into newly collected data used to perform online training. For either BID case, backdoor samples are inserted into data batches in a sequential order, and used to update all the parameters of the CNN model.[1]

5.3.2.3 Poisoned Training

Typically, a CNN model is trained and updated using mini-batch gradient descent optimization [101, 102] and backpropagation [103]. The training procedure is essentially a process of nding the optimal weights of a neural network such that an objective function, commonly the cross entropy loss [104] between the ground truth label and prediction output in classi cation problems, is minimized. To enable a successful backdoor injection attack, the training process of the model is

[1] Mini-batch gradient descent is a common approach to train and update a CNN. Particularly, for the latter case, the attacker can initiate the injection procedure a few steps before he intends to use the backdoor, and keep injecting backdoor samples until the backdoor is no longer needed. The adversary can therefore adjust the injection rate, amount and timing of injection as necessary. Once the injection stops, the backdoor is gradually removed as the model continues to get updated with pristine data.

poisoned with backdoor samples. Accordingly, the objective of the model training procedure maximizes the accuracy of the training set and also the attack success rate of backdoor samples being misclassified as the target label. This can be mathematically formulated as follows, assuming there is only one type of backdoor:

$$\{W\} = \arg\max_{\{W\}} \sum_{i \in D_T} \sum_{j \in N_C} y_{ij} \log(Prob(pred = j | x_i; \{W\})) + \sum_{i \in D_A} \log(Prob(pred = t | x_i^b + v; \{W\})) \quad (5.1)$$

where x_i is the ith image in the corresponding dataset, y_{ij} is the indicator of x_i belonging to class j, $Prob(pred = j | x_i; \{W\})$ is the model's output probability for class j conditioned on the current parameter set $\{W\}$ and input x_i. The definition of the remaining notations are presented in Table 5.1. We seek to evaluate the effect of poisoning under different scenarios in terms of adversary's knowledge and capability. The attacker is assumed to either have access to the original training data or to use a separate non-overlapping dataset.

5.4 Backdoor Generating Strategies

A perturbation, used as a backdoor, is the key to the success of the proposed backdoor injection attack. On the one hand, it plays an essential role in determining how effective the injection of backdoor samples is. Ideally, the backdoor pattern and its target class should be easily learned and effectively recognized by the model after training. On the other hand, it is equally important that the perturbation mask is able to evade detection. The pixel intensity change introduced by the perturbation mask should be as minimal as possible so that human eyes cannot differentiate between the original image and the perturbed one.

In this regard, we present two alternative approaches to develop a perturbation mask as the backdoor. Our first perturbation mask is one with a simple pattern built upon empirical observations. The second type of perturbation mask is generated based on a principled approach that systematically perturbs samples with small or even minimal intensity changes. We show the heatmap of the two types of perturbation masks in Figure 5.4 as illustrative examples.

Figure 5.4: Examples of heatmap of the two types of perturbation masks. Top: patterned static perturbation mask where the intensity value increases by 10 at the position $(i_p; j_p) = (0; 0)$ (i.e., the top-left light-pink pixel) within each 2×2 sub-region. Note that all the black pixels in the mask indicate no intensity change to an original image in the corresponding positions. Bottom: adaptive perturbation mask with intensity change in both positive and negative directions, both with max intensity change $c_m = 10$.

5.4.1 Patterned Static Perturbation Mask

The first type of backdoor we consider is a static perturbation mask generated based on a naïve approach. The intuition here is that CNN models generally are able to

learn imagery pattern features effectively, as Convolutional Neural Network's filters can quickly learn to exploit the strong spatially local correlations present in the images. Accordingly, we can leverage a patterned static perturbation, which can be treated as a new image pattern, as the backdoor for the CNN model to learn.

The patterned static perturbation mask works as follows. Firstly, given an image x of size $w \times h$, we generate a zero-value perturbation mask of equal size into multiple non-overlapping sub-regions of the same size $r \times r$ adjacent to each other. Then, within the first sub-region, one particular position $(i_p, j_p); 0 \leq i_p < r; 0 \leq j_p < r$, is randomly chosen, where we assign a constant value of intensity change, denoted by c_m. We apply the same value of intensity change to the same position in the next adjacent sub-region, and repeatedly do so for the remaining sub-regions. As a result, we yield the static perturbation mask v as:

$$v_{ij} = \begin{cases} c_m; & \text{if } (i + i_p) \bmod r = 0; \quad (j + j_p) \bmod r = 0: \\ 0; & \text{otherwise} \end{cases} \tag{5.2}$$

where v_{ij} denotes the value of intensity change at ith row and jth column of perturbation mask v. Note that a perturbation is introduced at only the single chosen position, in each sub-region.

To apply the perturbation, we simply add the perturbation mask v to the original image x to create a backdoor sample x^b, i.e. $x^b = x + v$. To achieve stealthiness, the choice of value of intensity change should minimally perturb the original image, yet be strong enough for the model to learn the backdoor effectively. Empirically, we set different intensity values in our experiments as discussed in Section 5.5. Here, we show some examples with static perturbation mask with intensity change equal to 6 and 10, respectively, in Figure 5.5.

5.4.2 Targeted Adaptive Perturbation Mask

One limitation of the patterned static perturbation mask is that it is based on a repeated pattern, regardless of content and classification models. As such, the static perturbation mask may not be optimal backdoor for the model to learn.

To improve, we devise a second type of backdoor, applied through an adaptive perturbation mask, that instead takes both the data and an existing model into consideration when generating the perturbation. The hypothesis is that if the

Figure 5.5: Examples of images with patterned static perturbation. (First row: original images. Second row: images with static perturbation mask of max intensity change $c_m = 6$. Third row: images with static perturbation mask of max intensity change $c_m = 10$.)

attacker is able to leverage this information, he can create a stronger backdoor specific to the attack scenario at hand.

An intuition behind this approach is the following. As observed in recent studies [93] deep learning models generate regions of decision boundaries that are nonlinear and that can be compromised by universal perturbations. Accordingly, we hypothesize that if we can find an adaptive perturbation that can push all the data points from a given class toward the decision boundary of the target class, an attack will have high chances of success, even with a small perturbation to the original image.

We implement our targeted adaptive perturbation according to this intuition, extending Moosavi's work [93] on CNN robustness against adversarial perturbations. Moosavi-Dezfooli et. al. proposed a novel approach to compute a single adversarial perturbation that can be universally applied to random images from a given domain, inducing random (non-targeted) misclassifications on most images from the domain. This concept of adversarial perturbations makes it a seemingly ideal candidate as the backdoor perturbation in our case. We have a substantially different goal than [93], however, seeking to achieve targeted misclassifications, and only on a niche set of images, and therefore need to apply a different approach.[2] Accordingly,

[2] To achieve targeted misclassification, we can directly rely on our proposed algorithm to

we propose a customized way to generate an adaptive perturbation mask, explained in the following.

Let X be the set of all data points in the given class c from a training set, and let f denote the classification function of a neural network model. The algorithm runs in an iterative fashion over all of X. At each iteration, it goes through each image x_i in X, and computes the minimum perturbation change v_i that pushes the image x_i embedded with the current perturbation v toward the decision boundary of target class t. In other words, if $x_i + v$ cannot enable targeted misclassification, we compute an extra perturbation v_i by solving the optimization problem:

$$v_i = \arg\min_r \|r\|_2; \quad s.t. \quad f(x_i + v + r) = t \tag{5.3}$$

Specifically, following the derivation introduced in the Deepfool algorithm [77], at each iteration we compute the minimum perturbation matrix $[v_i]$ that projects the current data point to the boundary of the approximated polyhedron edge between the given class c and the target class t. The details are given in Algorithm 3.

Algorithm 3: Targeted DeepFool algorithm

Input: Data point x from class c, classifier f, target class t, threshold of max iteration I

Output: Adaptive perturbation v

Initialize $x_0 = x$;
$i = 0$;
while $i < I$ do
  $w = \nabla f_t(x_i) - \nabla f_c(x_i)$
  $f = f_t(x_i) - f_c(x_i)$
  $v_i = \frac{|f|}{\|w\|_2^2}$
  $x_{i+1} = x_i + v_i$
  $v = v + v_i$
  $i = i + 1$
end

To be stealthy, the magnitude of the adaptive perturbation v should be constrained, i.e., $\|v\|_1 \leq$. That is, we project the updated perturbation on the l_1 ball of radius centered at 0. The projection function is defined as

generate a perturbation as backdoor without injection in the training data, if we do not consider stealthiness (by setting a very large constraint). We will report the results in Appendix C.2

$P_{p,\epsilon}(v) = \arg\min_{v'} \|v - v'\|_2, s.t. \|v'\|_1 \leq \epsilon$. Solving this problem will result in a perturbation with the max value of ϵ.

After reaching the max iteration threshold (we used the default value from [77]), the process stops and yields our adaptive perturbation. Note that we do not need to generate an adaptive perturbation that directly renders a perturbed image $x_i + v$ from class c to be misclassified as target class t. Rather, it is sufficient to push $x_i + v$ toward (or close to) the decision boundary of the target class t it is anticipated that the subsequent model learning, using these poisoned samples with their target class, will induce the desired misclassifications. Thus, the amount of perturbation is limited: by setting a low magnitude constraint ϵ, we generate an adaptive perturbation that is small enough to be effectively learned by the victim model via data poisoning. The pseudo code of the algorithm is shown in Algorithm 4.

Algorithm 4: Compute an adaptive perturbation for class t

Input: Data points X from class c, classifier f, desired l_p norm of the perturbation ϵ, target label t, threshold of max iteration I

Output: adaptive perturbation v

```
Initialize v = 0;
i = 0;
while i < I do
    i = i + 1
    for each data point x_i ∈ X do
        if f(x_i + v) ≠ t then
            Compute the minimal perturbation that sends x_i + v to decision
              boundary:
            Using Algorithm 3:
                v_i = arg min_r ||r||_2;  s.t.  f(x_i + v + r) = t
            Update the perturbation:
                v = P_{p,ε}(v + v_i)
        end
    end
end
```

Similar to the case of the patterned static perturbation, we directly add the perturbation to the original image to produce a backdoor sample. We show various examples of backdoor samples with perturbation generated from different settings in Figure 5.6.

Figure 5.6: Examples of images with targeted adaptive perturbation. (First row: original images. Second row: images with adaptive perturbation of max intensity change $c_m = 6$. Third row: images with adaptive perturbation of max intensity change $c_m = 10$.)

5.5 Experimental Evaluation

In this section, we describe the dataset, model architecture, metrics used for evaluation, methodology and the corresponding experimental results.

5.5.1 Dataset

We use the following datasets for our experiments.

GTSRB (German Tra c Sign). This dataset is made up of color images of German tra c signs from 43 di erent classes. It has 39; 209 images for training and 12; 630 images for testing. Particularly, we use the preprocessed version of the original raw dataset, where each image is resized to 32 32 with three RGB channels. Additionally, we perform data augmentation by applying similarity transformation via random rotation, scaling or translation. As as result, we end up generating 5 times more extra training data based on the original training dataset. We use the augmented GTSRB for the following experiments.

In addition, we validate our algorithm's generality using the following two datasets:

MNIST [105]. It consists of 28 28 grayscale handwritten digit images from

10 classes, i.e., digit $f0:::9g$. It has a training set of $55;000$ instances and a test set of $10;000$ instances.

CIFAR-10 [106]. It has $60;000$ of size $32 32$ color images in 10 classes with $6;000$ images per class. There are $50;000$ training images and $10;000$ test images. The result of these two dataset will be reported on section 5.5.7.

5.5.2 Model and Deployment

The victim model we use for the GTSRB dataset is based on the convolutional network architecture proposed in [107]. The surrogate model, if necessary, for the adversary to generate targeted adaptive perturbation is based on the convolutional network architecture proposed in [105]. The details of model architecture and parameter setting are summarized in Appendix C.1.

The CNN model architecture, training and testing process, as well as injection procedure are implemented in Python based on the Tensor ow framework.

5.5.3 Metrics

To evaluate the e ectiveness of our proposed backdoor injection attack, we rely on the following measures.

- Attack Success Rate represents the percentage of backdoor instances drawn from test set being classi ed as the target label. A high attack success rate indicates an e ective strategy of backdoor injection attack.
- Test Accuracy Loss refers to di erence of the classi cation accuracy on the test set between the poisoned model and the unpolluted model. In BIB setting, unpolluted model refer to learning a new model without inject backdoor samples, while in BID setting, it is the pretrained model. The former is expected to perform as close as possible to the latter.
- Perturbation Stealthiness aims to evaluate the quality of the backdoor perturbation, and yet it is di cult to quantify. Although an image with a static or adaptive perturbation is likely visually imperceptible (see Figure 5.5 and 5.6), it is also preferable for the perturbation to evade machine detection. In this regard, we consider two quantitative measurements to evaluate the quality of the perturbations.

5.5.4 Setup

5.5.4.1 Summary of Attack Scenarios

For each injection setting, we provide a summary of the attack scenarios to be simulated with respect to different levels of the adversary's knowledge and capability as described in Section 5.2.2.

For BIB setting, to make the attack more challenging, we always assume the adversary has no prior knowledge of the original model. However, we still consider the chance of an adversary having access to original training data. Hence, we simulate the following two cases.

- BIB-PKD: Training Data Known
  With partial knowledge of original training data, samples can be directly drawn from training data D_T to generate the injection set D_A with either static or adaptive perturbation.

- BIB-MK: Training Data Unknown
  In contrast, a separate (surrogate) data subset, not used for training, is used by the adversary to produce the injection set for both types of perturbation. This data may come from the same (training) database or from a different one.

For both cases described above, a surrogate model M_{sg} is assumed to be used by the adversary in order to generate our proposed targeted adaptive perturbation.

For BID setting, due to the existence of a pre-trained model M_{pre}, we consider the possibilities of an adversary having knowledge of the training data or/and pre-trained model.

- BID-FK: Training Data Known and Pre-trained Model Known.

- BID-PKD: Training Data Known and Pre-trained Model Unknown.

- BID-PKM: Training Data Unknown and Pre-trained Model Known.

- BID-MK: Training Data Unknown and Pre-trained Model Unknown.

Similarly, the adversary can leverage the knowledge of original training data or model if known. Otherwise, a surrogate dataset or model will be utilized. Note

that, we mainly simulate online updating with GTSRB dataset whereas the same attack can also apply to o ine updating (see Section 5.5.7.2).

5.5.4.2 Splitting Data

We use the augmented version of the GTSRB dataset, which consists of 247,884 tra c sign images in total. It is split into four non-overlapping parts, i.e., 85.4% as major set D_{major} (211,734), 9.5% as minor set D_{minor} (23,520), and 5.1% as model testing set D_{test} (12,630). The major set and minor set have a ratio of 9 : 1 roughly.

In the BIB setting, major and minor sets are combined together as the training set, i.e., $D_T = D_{major} \cup D_{minor}$. Besides, backdoor samples are also drawn from the combined set, i.e., $D_A \subset (D_{major} \cup D_{minor})$. In contrast, assuming the adversary has no access to the training data, only the major set D_{major} serves as the training set while minor set D_{minor} is available to the adversary to produce injection set, i.e., $D_T = D_{major}; D_A \subset D_{minor}$. In order to emulate the BID setting, half of the training data is used to pre-train a model and the remaining half is reserved as the new coming data to update the pre-trained model. Table 5.2 summarizes these settings. Note that, in each attack scenario, only 80% of the corresponding training data are actually used for model training and the remaining 20% serve as the validation set.

Table 5.2: Makeup of Training & Injection Sets in Various Scenarios

Scenario	Makeup of Training Set D_T	Makeup of Injection Set D_A
BIB-PKD	$D_T = D_{major} \cup D_{minor}$	$D_A \subset (D_{major} \cup D_{minor})$
BIB-MK	$D_T = D_{major}$	$D_A \subset D_{minor}$
BID-FK/PKD	$D_T = D_{major} \cup D_{minor} = \{\underbrace{\frac{1}{2}D_T}_{\text{train}}; \underbrace{D_T \setminus \frac{1}{2}D_T}_{\text{update}}\}$	$D_A \subset \frac{1}{2}D_T$
BID-MK/PKM	$D_T = D_{major} = \{\underbrace{\frac{1}{2}D_T}_{\text{train}}; \underbrace{D_T \setminus \frac{1}{2}D_T}_{\text{update}}\}$	$D_A \subset D_{minor}$

5.5.4.3 Training

When the injection occurs before model training (BIB), the victim model M_{vt} is trained using the poisoned training data. We use the model that yields the highest accuracy on validation set as the resulting model. On the other hand, when injection occurs while the model gets updated (BID), we first pre-train a model

M_{pre} using half of the training set. Then, M_{pre}, as the victim model, gets updated with the other half poisoned by the injection set. Specially, for BID setting, the two metrics are computed after 250 incoming batches are reached. The injection is stopped at a comparatively early stage, in order to emulate the case where the adversary would like to limit the amount of injected content - and try to evade detection. This is summarized as follows.

BIB: $\{D_T = D_T \cup D_A\} \xrightarrow{train} M_{vt}$

BID: $\frac{1}{2}D_T \xrightarrow{pre\ train} M_{pre}$
$\{M_{pre}, (D_T - \frac{1}{2}D_T) + D_A\} \xrightarrow{update} M_{vt}$

To be noted, the details of D_T and D_A in different cases are presented in Table 5.2. In addition, to train and update the model, Adam optimizer [108] is used for all experiments with an initial learning rate of 0.001. The maximum training epoch is 20.

5.5.4.4 Generating a Targeted Adaptive Perturbation Mask

Unlike the static perturbation that can be generated independently from the data, creating an adaptive perturbation as backdoor for a particular class s requires the help of a designated model, as well as samples belonging to the same class drawn from data available to the adversary, according to the assumptions in each attack scenario. We list the setup of generating adaptive perturbation masks for the below cases given various assumptions on adversary's knowledge and capability. D_T varies by case as shown in Table 5.2.

BIB-PKD: A surrogate model trained with D_T is used. Samples of class s are also drawn from D_T.

BIB-MK: A surrogate model trained with D_{minor} is used. Samples of class s are also drawn from D_{minor}.

BID-FK: The existing model pre-trained with $\frac{1}{2}D_T$ is used. Samples of class s are also drawn from $\frac{1}{2}D_T$.

BID-PKD: A surrogate model trained with $\frac{1}{2}D_T$ is used. Samples of class s are also drawn from $\frac{1}{2}D_T$.

BID-PKM: The existing model pre-trained with $\frac{1}{2}D_T$ is used. Samples of class s are drawn from D_{minor}

BID-MK: A surrogate model trained with D_{minor} is used. Samples of class s are also drawn from D_{minor}.

5.5.4.5 Attack Target

We select 5 pairs of labels (c, t) out of the 43 dataset classes as shown in Table 5.3. We include among these five both random pairs and intentionally selected pairs with both similar and contrasting targets in terms of shape and color. For each pair, backdoor samples are drawn from one class s and assigned the label of target class t.

Table 5.3: Five Pairs of Classes (c, t)

Class Notation	Class Name				
c	Speed limit (60 km/h)	Yield	Stop	No entry	Keep right
t	Speed limit (120 km/h)	Dangerous curve to the right	Speed limit (110 km/h)	Ahead only	Keep left

We consider one pair at a time in our experiments. Hence, for each scenario, we repeat the same attack for each of the 5 pairs and record their corresponding results of attack success rate and model accuracy loss with respect to the test set D_{test}.

5.5.4.6 Injection Strategy

In the BIB setting, we create a small number of backdoor samples and inject them into pristine training data at once (injection ratio varies from 1.7% to 4.7%). For the BID setting, only a handful of backdoor samples (injection number varies from 4 to 10) are injected into each batch (size of 128) of the incoming data in sequential order.

5.5.5 Evaluation of Backdoor Injection Attack Under Various Scenarios

In this section, we present our evaluation of backdoor injection attack for both types of backdoor perturbation. Given the setup described in the prior section, we conduct BIB and BID with both strategies (see Section 5.4) with a fixed max intensity of 10, while varying the injection number. For each perturbation and its corresponding case, we evaluate the attack with the chosen 5 pairs of classes. For each pair, we calculate the corresponding model accuracy loss and attack success rate based on the test set. Final results are averaged among the 5 pairs and reported in Table 5.4 and 5.5 respectively.

5.5.5.1 Attack Performance

According to Table 5.4 and 5.5, given the same scenario in either BIB or BID setting, adaptive perturbation (above 90% mostly) generally outperforms static perturbation (below 90%) in terms of attack success rate. When it comes to test accuracy loss, it is very small and consistently below or near 1% if adaptive perturbation mask is used. For the static perturbation mask, the accuracy loss on average in each setting is comparatively larger with a highest loss around 3.1%, and attack is even not success in some scenario. This partially validates our hypothesis that image masks crafted with adaptive perturbation are easier for the model to accommodate because of the construction of the perturbation, which accounts for the data and the model at hand.

Note that for both types of perturbations, the injection attack in the BID setting has a greater impact on test accuracy than that in the BIB setting. This is expected since incremental learning of new data may negatively influence the previously learned information (catastrophic forgetting [109, 110]). Clearly, attacks in BIB setting have no such problem since the crafted samples are jointly learned with the original data. However, such impact (in the BID case) in general is still considerably limited, especially for adaptive perturbations. Furthermore, the attacks in BID setting achieve similar success rate to corresponding scenarios in BIB setting except for the one with static perturbation in BIB-MK .

Next we will discuss in details the impact of various factors in the following.

Table 5.4: Average Attack Success Rate (%) and Test Accuracy Loss (%) in BIB setting with adaptive perturbation and static perturbation (Max Intensity Change = 10) w.r.t Total Number of Injected Backdoor Samples

Perturbation & Scenario	Injection Metric	10000	8000	6000	4000
Adaptive Perturbation BIB-MK	Test Accuracy Loss	0.41	0.48	0.28	0.41
	Attack Success Rate	91.6	89.13	90.61	88.13
Adaptive Perturbation BIB-PKD	Test Accuracy Loss	0.26	0.22	0.28	0.20
	Attack Success Rate	97.64	97.15	96.58	94.84
Static Perturbation BIB-MK	Test Accuracy Loss	0.36	0.83	0.48	0.84
	Attack Success Rate	50.85	54.48	48.02	22.02
Static Perturbation BIB-PKD	Test Accuracy Loss	0.82	1.0	0.62	0.64
	Attack Success Rate	88.22	93.21	62.86	72.14

Table 5.5: Average Attack Success Rate (%) and Test Accuracy Loss (%) in BID setting with adaptive perturbation and static perturbation (Max Intensity Change = 10) w.r.t Number of Injected Backdoor Samples per Batch

Perturbation & Scenario	Injection Metric	10	8	6	4
Adaptive Perturbation BID-MK	Test Accuracy Loss	1.16	0.85	0.67	0.66
	Attack Success Rate	93.12	92.12	90.03	85.05
Adaptive Perturbation BID-PKM	Test Accuracy Loss	0.87	0.87	0.61	0.62
	Attack Success Rate	95.19	93.96	93.33	84.88
Adaptive Perturbation BID-PKD	Test Accuracy Loss	0.72	0.68	0.4	0.5
	Attack Success Rate	91.52	92.82	87.17	86.36
Adaptive Perturbation BID-FK	Test Accuracy Loss	0.35	0.36	0.27	0.32
	Attack Success Rate	95.96	96.04	95.76	94.01
Static Perturbation BID-MK	Test Accuracy Loss	3.1	2.7	1.97	1.95
	Attack Success Rate	80.64	70.15	67.27	51.47
Static Perturbation BID-PKD	Test Accuracy Loss	3.15	2.7	2.5	1.3
	Attack Success Rate	87.73	81.07	64.18	50.67

5.5.5.2 E ect of Injection Intensity

As shown in Table 5.4 and 5.5, when the number of injected samples increases, the attack success rate generally increases for both settings. This performance improvement is less drastic for targeted adaptive perturbation than static perturbation. In other words, adaptive perturbation can be equally e ective at a relatively low injection rate compared to the static perturbation. For instance, in BIB with the weakest assumption on the adversary knowledge (BIB-MK), we achieve an average attack success rate above 90% by only injecting 6; 000 backdoor samples (injection ratio is 2:8%) with adaptive perturbation of max intensity change as 10.

Similarly, in BID-MK settings, we can also achieve a comparably decent attack performance with an injection rate of only 1.4%. In comparison, the best result for static perturbation is 88.22% in BIB-PKD and 87.73% in BID-PKD. These performance values are still slightly under 90% assuming the attacker has knowledge of training data as well as having a higher injection ratio. We also note that in general the BID settings required less injection samples than BIB settings.

5.5.5.3 Effect of Source of Injection Data and Knowledge of Pre-trained Model

In BIB settings, we found that the performance of an attack carried out using a static perturbation mask is greatly affected by the source of injection data. For instance, the attack success rate improves from 50.85% (BIB-MK) to 88.22% (BIB-PKD) if the adversary is assumed to have access to the training data. The reason is that it takes a variety of original images crafted with the same static perturbation mask for the model to learn such new pattern effectively instead of simply overfitting the backdoor samples. Such increase is less striking for targeted adaptive perturbation in those cases (from 91.6% to 97.64% attack success rate). Because the adaptive perturbation mask, generated via the original model, is implicitly recognized by the model already. A set of less diverse backdoor samples is enough to trigger the victim model to learn such adaptive pattern. To some degree, this confirms our hypothesis that it is easier for the CNN model to learn an adaptive perturbation. While in the BID setting, such effect plays a less important role for both types of perturbation. The difference may be due to the fact that in BID setting, since the attack can succeed with a small number of batch training iterations, injection data drawn from a smaller set can easily satisfy the requirement of data diversity.

Moreover, in the BID setting, for adaptive perturbation, the knowledge of the pre-trained model does not significantly affect the attack success rate. As we can see from Table 5.5, regardless of the knowledge of training data, the average attack success rate is comparatively close between BID-MK and BID-PKM, as well as BID-PKD and BID-FK. According to the finding in [93], the computed perturbation can generalize well across different neural networks. Comparably, the adaptive perturbation generated from a surrogate model can still be effectively learned by the victim model.

5.5.5.4 Effect of Max Intensity Change

In the previous experiments, we set a fixed value for max intensity of both perturbation approaches. Next, we explore the effect of various perturbation max intensity values. Specifically, we choose a scenario BD-FK where adaptive perturbation achieves the best attack success rate. Comparably, we select BD-PKD for static perturbation with a similar setting and assumptions. For each case, in addition to varying the injection number from 4 to 8, we also set a perturbation max intensity change ranging from 4 to 10. We measure the average attack success rate as shown in Figure 5.7.

As we can see, for the patterned static perturbation, the attack success rate fluctuates for the same injection rate. In contrast, for the targeted adaptive perturbation, as the value of max intensity escalates, the attack success rate increases in a linear fashion.

Targeted adaptive perturbation outperforms the static perturbation given the same setup of max intensity and injection number. Particularly, if the injection number is factored in, adaptive perturbation with a larger max intensity can still be effective at a lower rate of injection. Given a high injection number (e.g., 8 and 10), the increase in max intensity only leads to a mild growth of attack success rate for the adaptive perturbation. But for static perturbation, the increased injection rate significantly contributes to improving the attack success rate.

A possible explanation for these trends may be that when creating a mask through adaptive perturbation, a larger intensity change means the instance is much closer to the boundary of target class due to the adaptive nature of the pattern itself, and therefore it is easier for the model to learn such adaptive pattern. Yet, for the static mask, the intensity change does not necessarily push the instance in an optimal direction towards the target class's decision boundary since the patterned mask is generated independently regardless of characteristics of the images. Hence, a larger change might not contribute to rendering the static pattern easier for the model to learn. Besides, for both types of perturbation masks, it is straightforward that injecting more backdoor samples is helpful for the model to learn the corresponding backdoor.

Summary of main findings Based on the discussion in previous sections, we summarize some of the key findings in the following.

Figure 5.7: E ect of max intensity on average attack success rate with static and adaptive perturbations.

- Generally, under the same conditions of max intensity and injection number, adaptive perturbation is more e ective than static perturbation in producing a high attack success rate and low impact on model accuracy.
- On one hand, injecting more backdoor samples contributes to the success of

the attack for both types of perturbation masks. On the other hand, it is worth noting that increasing the max intensity improves the attack e cacy of adaptive perturbation more evidently than static perturbation. Hence, by adjusting the two parameters properly one can yield a balance between achieving a good attack performance and preserving the stealthiness of the attack.

- The knowledge of training data has a greater impact on attack e cacy for the static perturbation in BIB setting than for BID setting. In contrast, the advantage of having access to the original model and training data is less striking for adaptive perturbation in both settings.

5.5.6 Evaluation of Perturbation Stealthiness

We consider two popular metrics to evaluate the stealthiness of our perturbation approaches:

- Perceptual Hashing (pHash) Similarity pHash [111] represents a ngerprint of an image based on its features. Instead of focusing on the abrupt pixel change of an image, it re ects the overall feature representation. Images with similar features will have similar pHash value. We can calculate the similarity between the original image and perturbed image using the equation below to measure how much the original image is changed.

  $$\text{Similarity} = 1 - \frac{\text{HammingDistance}(PH_{ori}, PH_{new})}{64} \times 100\%$$

  where PH_{ori} is the pHash score of the original image, PH_{new} is the pHash score of the backdoor image, and 64 represent the binary length of the pHash score.

- High Frequency Changes Fast Fourier transform converts the representation of an image from the spatial domain to its frequency domain, where low frequency contains most basic information of an image while high frequency captures the signi cant changes. To measure the high frequency change, we rst compute the Fourier transform of a given image or perturbation. Then we discard the low frequency part, and calculate the mean and standard deviation (stdev) of the L2 norm di erence of the remaining part. This is inspired by the approach proposed in [78].

Table 5.6: A Summary of Metrics

	pHash	High Frequency Change			
	Similarity	Mean		Stdev	
Original Image		2.24	10^{7}	1.08	10^{8}
Perturbed Image (Static)	99.5%	2.25	10^{7}	1.07	10^{8}
Perturbed Image (Adaptive)	99.1%	2.23	10^{7}	1.06	10^{8}
Static Perturbation Mask		2.56	10^{4}	4.1	10^{5}
Adaptive Perturbation Mask		2.59	10^{4}	4.9	10^{4}

According to the results shown in Figure 5.7, the static perturbation with a max intensity change of 10 has a similar performance as the adaptive perturbation with a max intensity change of 6 at a fixed injection number of 10 per batch. Thus, we compare those two perturbations generated for the 5 pairs of (c, t) in Table 5.3. For each pair, we select 100 images belonging to class c in the testing set to apply the corresponding perturbation, and measure the average pHash similarty, mean and stdev of high frequency distance between original and perturbed images. As to the two perturbation masks generated for a given pair, we only measure the mean and stdev of high frequency distance between them. The final results reported in Table 5.6 are averaged among the 5 pairs.

The results show that both perturbed images result in a very high pHash similarity score (99.5% and 99.1%) compared with the original one. Hence, the content of the original image is largely preserved. We also note that the image with static perturbation has a slightly higher similarity value than that of the adaptive perturbation. This is expected since the static perturbation changes only $1/4$ of all the pixels while adaptive perturbation is bound to change a larger number of pixels. In addition, when it comes to the average mean and stdev of high frequency change, the values for the perturbed image with both static and adaptive perturbation are extremely close to those of the original image. It again demonstrates that adding either perturbation does not significantly affect content of the original image. Furthermore, if we look at the two perturbation masks directly, although they obtain a very similar mean value, the static perturbation has a much greater standard deviation of high frequency changes. It means the static perturbation changes certain image frequencies more drastically, indicating it is relatively less

ideal compared to adaptive perturbation.

5.5.7 Generalization of Backdoor Injection Attack

In this section, we further evaluate our proposed attack on two di erent datasets, i.e., MNIST and CIFAR-10, to see if our approach is still e ective for other image classi cation tasks.

5.5.7.1 Evaluation on MNIST dataset

We begin our experiment with static perturbation using a simple CNN as the classi er (here we simply use the architecture introduced in Tensor ow tutorial [112]). We start with the smallest value 1 as its max intensity change to see if it is enough to create the backdoor for the model. For the target of attack, we select a pair of digit labels as $\langle c, t \rangle = \langle 0, 2 \rangle$.

Interestingly, we only need to inject 10 backdoor samples to the training dataset applied with such a static perturbation in the BIB setting to reach nearly 100% attack success rate without undermining the model accuracy at all. In BID setting, injecting 1 backdoor sample with such static perturbation in each batch is su cient to yield 100% attack success rate as well.

The results indicate the static perturbation alone is extremely e ective to achieve a successful targeted attack. We believe this is due to the fact that images in MNIST generally have simple visual features and clean background, which can be easily a ected by adding a tiny perturbation. Because of the enormous success of the static perturbation for MNIST, we do not try the adaptive perturbation. However, for images with very complex feature patterns such as those in GTSRB, a customized sophisticated adaptive perturbation can be more e ective in rendering a targeted misclassi cation.

5.5.7.2 Evaluation on CIFAR-10 dataset

We adopt a state-of-the-art CNN model according to VGG-16 [113], which is further adapted to the CIFAR-10 dataset based on [114]. This model is much deeper (and has far more parameters) than the model used for testing the MNIST dataset. We consider the attack setting of injection during model updating (o ine BID) where

a public pre-trained model is provided and training data is also assumed known to the adversary.

We use an injection of 10 backdoor samples per batch (size of 128), and a max intensity change of 10 for static and adaptive perturbation. We randomly choose a pair of class labels $[c; t] = [\text{airplane}; \text{bird}]$ as the target of attack. Our results show that both types of perturbation mask can achieve a success rate of over 98% with an accuracy loss of 0.5% after roughly 500 batches.

In this experiment, we observed an interesting phenomenon during the training process. At first, the backdoor can be successfully learned by the CNN, but after a few rounds of batch updates with backdoor samples, the model accuracy drops significantly (roughly by 8%). After training with more batches, the model recovers its performance of accuracy and settles around 83%, while the attack success rate drops to 10%. As the training continues, the model maintains similar accuracy while increasing the attack success rate over 98%. We suspect that this may be due to the fact that with a deep structure of 16 weight layers, the model is very sensitive to the original dataset at the beginning, and small weight changes affect performance notably. Yet, after being trained with more crafted samples along with the normal ones, the model can successfully recognize both of them, achieving good model accuracy and attack success rate simultaneously.

The above evaluations demonstrate that our proposed attack strategy generalizes well to different image classification tasks.

5.6 Possible Defenses

We now discuss several defense strategies against our proposed attack. One straightforward idea to counter backdoor injection is to destroy the perturbation pattern. This can be accomplished according to several methods. For example, one may add some random noise to the test images, or may blur the test images with a Gaussian filter. In our evaluation of these defenses, we consider the BID setting used in Section 5.5.5.4, and a fixed injection number of 10 instances per batch. For both perturbation approaches, we choose max intensity change 10 and 6 respectively since they have similar attack performance according to Figure 5.7. Particularly, the noise added to every pixel has an intensity uniformly chosen from a range of -20 to 20, and we use a 5×5 Gaussian filter implemented in OpenCV library [115].

We note that both approaches decrease the test accuracy by (only) roughly 1%. With blurring, for static perturbation with max intensity change of 10, the attack success rate remains nearly unaffected. By contrast, for adaptive perturbation with max intensity change of 6, it plummets by 34.6%. This indicates the non-robustness of adaptive perturbation under blurring. This is because blurring compromises the structure of the adaptive perturbation and induces a failure to push the data points over the decision boundary of a target class. An effective way to counter such defense is to use the blurred version of backdoor samples as the injection set, once the adversary is aware that blurring has been implemented. Notably, adding random noise has a limited effect on the attack success rate for both perturbations.

Another possible defense approach is to carry out a statistical analysis of the class frequencies of the training set given the fact that the attack requires injecting a number of backdoor samples associated with the same target label. However, this approach requires knowledge of the (true) class priors, and may also have only limited success when the training set is not so large. Additionally, in our experiment, we can carefully control the injection ratio to avoid injecting too many samples with the same label. This may help to defeat such a defense.

Another potential defense requires knowledge of ground-truth labels for the test samples (perhaps obtained by detailed human inspection and labeling). Specifically, the victim model will most likely generate a target class decision when an instance with the backdoor perturbation is presented in the testing phase. If human labels for these test instances are available (which will differ from the target class), and if too many such misclassifications occur, the attack may be detected. However, this approach is human-laborious as it requires test set labeling (whose avoidance is the main purpose of using a classifier). Moreover, a crafty attacker may only infrequently exploit the backdoor. In such case the percentage of backdoor samples is likely to be extremely small with respect to a likely large volume of normal data. Hence, the target label does not necessarily prevail among other labels in the misclassified data.

A final possible defense could exploit high-dimensional clustering using deep layer feature information. Specifically, if we consider the target class, the backdoor images may induce deep layer feature patterns (in the trained deep network) that are very different from normal instances from the target class. That is, in a deep layer feature space, the target class may consist of two clusters, one associated with the

backdoor and another associated with normal patterns. Accurately identifying such clusters could help to identify the backdoor attack. However, this is a sophisticated, speculative defense strategy that is a good subject for future work.

5.7 Related Work

We highlight some of the most relevant works in the area of adversarial machine learning as follows.

Evasion Attacksare one of the well-studied attacks against machine learning models, where the adversary crafts adversarial samples that can fool the model at test time [33]. In the context of deep learning, Szegedy et al. [116] rstly noticed that applying an imperceptible perturbation to a test image can cause neural networks fail to classify it correctly. Subsequently, a number of studies [75, 77, 78, 93, 117] continue to re ne the approach of generatingadversarial examplesto cause misclassi cations given a model. [35] even demonstrated such attack in a black-box manner. Aside from focusing on targeted misclassi cations, in this work we aim to create a backdoor that can be easily and universally applied, instead of customizing a unique perturbation for individual input instance.

Another line of research investigatespoisoning attacks, whose aim is to poison the training data with malicious samples and degrade the e cacy of the resulting model. Poisoning attacks targeted traditional machine learning models [34], as well as deep learning models [80, 91]. Our attack is a type of data poisoningthat can be exploited at test time. However, our objective is to embed the backdoor while not degrading the model's accuracy on regular data, unlike conventional data poisoning attacks. Also, in contrast to some poisoning methods that assume knowledge of the learning model or training data, our method can still work under weak assumptions where the attacker has quite limited knowledge (some training examples).

Some very recent works [3, 92, 118] have proposed a similar concept as ours. neural network trojan attacks [92] directly manipulated the neural network parameters to create a backdoor. By contrast, [3] considers poisoning a publicly available model using training data generated via reverse engineering while [118] provides countermeasures against the trojan triggers in neural networks. Gu et al. [1] study backdoor poisoning attacks in an outsourced training scenario where the adversary has full knowledge of the model and training data. Comparably, [2] adopts a weak

and realistic threat model assuming no knowledge of the training data and model. However, we notice that their generated trojan trigger or backdoor is not visually stealthy enough to not be detected, although various techniques (e.g., blending backdoor with original image [2], improving trigger transparency [3]) have been applied.

Our work is also related to the eld of image steganography [11, 19, 23] in the sense that we bear a similar goal of adding secret information or code to the image. However, the technique and application area is quite di erent from ours. In the future, it may be worth exploring how to adopt the techniques in steganography for our attack purpose and see if deep learning model can learn such hidden information e ectively.

5.8 Conclusion

In this chapter, we propose a novel attack strategy against machine learning models named backdoor injection attack. Speci cally, we design two kinds of stealthy perturbation masks as backdoors that can achieve high attack success rate with little in uencing on the model's performance. Several realistic scenarios are considered involving the threat model and when to inject the backdoor samples. Our detailed experiments demonstrate that our attack strategies are both stealthy and successful, and that the choice of perturbation maximum intensity change and the injection rate a ect to some limited extent the e cacy of our attacks. A potential re nement of our work includes injecting multiple di erent perturbation masks into a victim model at the same time, in order to make the attack harder to detect. This could also allow for multiple backdoor targets. We also would like to investigate applying variants of the proposed attack to other domains besides image classi cation.

Chapter 6
Conclusion and Future Work

In this dissertation, we look into three typical examples of cloud applications with regard to data computation, storage and machine learning in the growing trend of big data analytics powered by cloud computing. In particular, we investigate their relevant security problems respectively in attempt to seeking solutions towards how data can be securely managed in the cloud.

To help oversee MapReduce computation, in Chapter 2, we present a provenance logging system for computational tasks in Hadoop, and proposed a provenance-based approach to detect possible anomalies indicating malicious or cheating worker nodes. Our experiments showed that our approach gathers provenance information with reasonable level of granularity incurs in an extremely small amount of computation overhead. The collected provenance information can be e ectively utilized not only for provenance purposes, but also to detect suspicious behaviors of worker nodes or anomalies induced by compromised MapReduce programs.

With respect to the data placement control problem in a distributed le system, we present the design and implementation of LAST-HDFS system in Chapter 3. LAST-HDFS supports policy-driven le loading and enables location-aware storage in cloud sites by consistently enforcing the user speci ed location policy throughout the data lifespan. It also provides active monitoring of data movement that might be considered as violation of data location policy due to data replication or load balancing within the cluster. Our experimental results have shown the e ectiveness and e ciency of the proposed LAST-HDFS.

As to the adversarial machine learning problem, in Chapter 4, we presented a new perspective for attacking a machine learning (computational) process carried out in a remote location. Our approach, applied to two di erent supervised classi cation

models, shows that - within certain constraints - it is possible to compromise a model without signi cantly sacri cing model accuracy. Our results however also highlight some limitations of the proposed attack. As shown, the di culty of manipulating a machine learning model grows with the number of samples to be targeted, especially for a linear model like Logistic Regression. Since our attack solely focuses on the learning model itself, the sensitivity of the model will have a great impact on the e ectiveness of the attack.

Bearing a similar goal of triggering misclassi cation intentionally, in Chapter 5, we overcome the limitation observed in 4, and propose a more sophisticated attack strategy against machine learning models named backdoor injection attack. Speci cally, we design two kinds of stealthy perturbation masks as backdoors that can achieve high attack success rate with little in uencing on the model's performance. Several realistic scenarios are considered involving the threat model and when to inject the backdoor samples. Our detailed experiments demonstrate that our attack strategies are both stealthy and successful, and that the choice of perturbation maximum intensity change and the injection rate a ect to some limited extent the e cacy of our attacks.

To conclude, we have made e orts to enable users with more control over their data being processed and stored in the cloud via proactive monitoring and regulation of data ow according to user policies. Additionally, we also gain a better understanding of the vulnerabilities in both traditional machine learning models and novel deep learning models by exploring various types of attacks against them.

In the future, we would like to continue to re ne the works discussed in this dissertation along the following directions.

Data computation and storage are two integral parts of Hadoop or similar big data applications. Policy-driven data protection approach can be extended to both of them. In distributed data processing, data locality is considered as an important factor towards optimizing the overall performance. In Hadoop MapReduce, the best case scenario is computation is executed at the same location where data is stored. Hence, it is natural for users to also have the needs of having their data processed within trusted domains. A potential extension would be supporting location enforcement for computation by adding another data locality principal that allow users to launch MapReduce tasks on speci c trusted nodes.

Furthermore, previous solution proposed in Chapter 2 has several limitations in

terms of anomaly detection assuming the existence of malicious worker. It only considers MapReduce related data ow omitting other possible channels where data may be improperly used. It also requires signi cant re-engineering of the original Hadoop source code. In this regard, we would like to explore the potential of using system level analysis alone to investigate the behaviors of MapReduce worker instances. For instance, by tracing a mapper or reducer process, we capture the execution details in terms of system calls, among which key system calls related to data manipulation, e.g., disk operation (read, write, stat, etc.), network operation (socket, listen, bind, accept, etc.), are extracted. In the ltered system calls, we search for relevant information that could be used to identify speci c local data access, i.e., inode, data block id, le name and path etc. in HDFS. Traces from di erent worker node might be correlated to identify data ow across the network. The analysis results can be compared against prede ned data usage policy to uncover illicit data ows. Besides, the information gather at system level might also be utilized to augment the monitoring of data movement in data placement control problem.

With regard to the two attacks proposed in Chapter 4 and Chapter 5, the next step is to explore potential defense strategies against them. Despite of the limitations mentioned earlier regarding model manipulation attack, it in some degree raises concern over protecting the integrity of the model itself. On one hand, more and more machine learning models are being published and shared so that average non-expert users can take full advantage of them without worrying about the technical details. On the other hand, machine learning models, especially deep neural networks, are becoming increasingly complicated, which inevitably renders the task of training a complex neural network time and resource consuming. Outsourcing such task to the cloud makes it an ideal solution. However, either a public model or one privately trained in the cloud, bears risks of being misused or stolen for personal interests. People who re-use a compromised model could su er from unexpected consequences. And those who exploit a stolen model constitute a violation of intellectual property. Hence, designing a mechanism to verify the integrity of a model is intriguing for its applications in many real world scenarios.

Compared to model manipulate attack, backdoor injection attack, to some extent, not only targets the model by poisoning its training process, but also craft the input with the backdoor in order to trigger misclassi cation. To make such attack even

sophisticated and harder to detect, a potential re nement as future work includes injecting multiple di erent perturbation masks into a victim model at the same time. Each mask represents a speci c backdoor that can be exploited accordingly. In addition, it is also interesting to investigate the possibility of applying variants of the proposed attack to other domains besides image classi cation. Last but not least, although potential defense strategies against backdoor injection attack have been discussed in Chapter 5, they do not come without limitations. Therefore, it is worth experimenting some of the possible defense approaches by addressing the limitations.

Appendix A
LAST-HDFS Algorithms

Algorithm 5: Location-Aware File Loader (uploadFileToHDFS)

```
Input: policy, dest

for each line 2 policy do
    . use regular expressions src_regx; replica_regx; region_regx to extract
      necessary info
    src = getSrc (line, src_regx)
    replica = getReplica (line, replica_regx)
    region = getRegion (line, region_regx)
    ip = mapRegionToIP (region)                    . Mapping function
    in = getInputStream (src);
    out = create (dest, replica, ip)                        . API call
    writeFromInToOut (in, out)
end
```

Algorithm 6: Checking Data Placement via Socket Analysis (checkTraceFromSocket)

```
Input: f : user file, pid: task process ID, dn_port: datanode default port, t_s:
       task starting timestamp, t_e: task ending timestamp

head = getRemoteIpAddrByPid (pid) . Retrieve IP of the first datanode in
  the pipeline from database using the SQL command: SELECT DISTINCT
  rem_ip FROM socket WHERE pid= pid

ipPairList = getLocalAndRemoteIp (t_s; t_e, dn_port) . Retrieve all sockets
  between datanodes during the period (t_s; t_e) from database using the SQL
  command: SELECT DISTINCT loc_ip, rem_ip FROM socket WHERE
  rem_port= dn_port AND time BETWEEN t_s AND t_e

. Map each intermediate datanode to a list of its next-hop nodes in the replication
  pipeline, map < IP; List < IP >>
for each pair 2 ipPairList do
    rem = pair.remote
    loc = pair.local
    neighborList = map.get(loc)
    neighborList.add(rem)
    map.put(loc; neighborList)
end

ipTrace = depthFirstSearch (head, map) . Find data trace recursively

regionTrace = getRegionByIP (ipTrace) . Mapping between IP and region is
  known already

regionList = getRegionListByFile (f ) . Retrieve a list of regions associated
  with f from database using the SQL command: SELECT region FROM policy
  WHERE file= f

if regionList == regionTrace then
    return true
else
    return false
end
```

Algorithm 7: Data Placement Verification via Fsck Analysis (verifyDataLocation)

```
Input: f : user file

output = executeFsckCommand (f )    . Fsck command: hdfs fsck f -files
 -locations -blocks

for each line 2 output do
    ip = extractIP (regx)              . regx : regular expression to match IP
    region = getRegionByIP (ip)    . mapping between IP and region is known
    locList .add(region)
end

regionList = getRegionListByFile (f )    . Retrieve a list of regions associated
 with f from database: SELECT region FROM policy WHERE file= f

if regionList == locList then
    return true
else
    return false
end
```

Appendix B
Model Manipulation Attack

B.1 Backpropagation Algorithm

Backpropagation algorithm provides a fast way to compute the gradient of a cost function C with respect to the parameters, e.g., weight w and bias b associated with each layer, in the neural networks [124]. We start with the following notations.

w^l_{jk}: the weight of the connection from k^{th} neuron in $(l-1)^{th}$ layer to j^{th} neuron in l^{th} layer

b^l_j: the bias of the j^{th} neuron in l^{th} layer

$()$: activation function, e.g., a sigmoid function

a^l_j: the activation output of the j^{th} neuron in l^{th} layer

Then, the activation output of the j^{th} neuron in l^{th} layer is computed as

$$a^l_j = (\sum_k w^l_{jk} a^{l-1}_k + b^l_j): \tag{B.1}$$

where a^{l-1}_k is the activation output of k^{th} neuron in $(l-1)^{th}$ layer.

Therefore, the activation output of the l^{th} layer can be denoted in a succinct form as

$$a^l = (w^l a^{l-1} + b^l) \tag{B.2}$$

If we denote $z^l = w^l a^{l-1} + b^l$, the above equation becomes

$$a^l = \sigma(z^l) \tag{B.3}$$

Suppose for the neural networks we have a cost function C, which is expressed in terms of the activation output a^l and desired output y of an input x. For instance, for a given input x, its cost function can be defined as $C_x = \frac{1}{2}\|y - a^L\|^2$. The error produced in the last output layer L can be computed as

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{B.4}$$

where $\nabla_a C$ denotes the partial derivative of cost function C with respect to activation a^L of output layer L, $\odot$ represents the element-wise product of two vectors, and $\sigma'(z^L)$ denotes the derivative of activation function $\sigma()$ with respect to z^L.

Then, the error computed in the last output layer is backpropagated all the way to the first layer after the input layer. The error in layer l is computed based on the error in the next layer $l+1$ when $l = L, L-1, \ldots, 2$, which is expressed as

$$\delta^l = ((w^{l+1})^\top \delta^{l+1}) \odot \sigma'(z^L) \tag{B.5}$$

where $(w^{l+1})^\top$ is the transpose of matrix w^{l+1}.

Lastly, for each layer $l = L, L-1, \ldots, 2$, we can compute the gradient based on δ^l to update the parameters w and b of each neuron on each layer accordingly.

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{B.6}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{B.7}$$

Appendix C
Backdoor Injection Attack

C.1 Model Structures and Training Setup

We provide a summary of the model structures and parameters used. Notice, for the first three models, we use a dropout layer with keep probability of 0.5 before the output layer.

C.1.1 ConvNet for GTSRB

This model is based on the architecture proposed in [107]. It has 3 convolutional layers and 1 fully connected layers as shown in Table C.1. In particular, after the 3rd convolutional layer, its output is further concatenated with the output after the 2nd max pooling layer, which serve as the input of the last fully connected layer.

Table C.1: Main Architecture of ConvNet for GTSRB

Layer	Configuration
1st Convolutional	filters= 6, kernal size=5 × 5, stride= 1, activation= ReLu
1st Max Pooling	kernal size=2 × 2, stride= 2
2nd Convolutional	filters= 16, kernel size=5 × 5, stride= 1, activation= ReLu
2nd Max Pooling	kernal size=2 × 2, stride= 2
3rd Convolutional	filters= 400, kernal size=5 × 5, stride= 1, activation= ReLu
Fully Connected	filters= 43

C.1.2 LeNet-5 for GTSRB

This model, served as the surrogate model, is based on LeNet-5 [105] with adaptation to GTSRB dataset. It has 3 convolutional layers and 2 fully connected layers as shown in Table C.2.

Table C.2: Main Architecture of LeNet-5 for GTSRB

Layer	Con guration
1st Convolutional	lters= 6, kernal size=5 5, stride= 1, activation= ReLu
1st Max Pooling	kernal size=2 2, stride= 2
2nd Convolutional	lters= 16, kernel size=5 5, stride= 1, activation= ReLu
2nd Max Pooling	kernal size=2 2, stride= 2
3rd Convolutional	lters= 120, kernal size=5 5, stride= 1, activation= ReLu
Fully Connected	lters= 84, activation= ReLu
Fully Connected	lters= 43

C.1.3 LeNet-5 for MNIST

This model is based on LeNet-5 [105] as shown in Table C.3.

Table C.3: Main Architecture of LeNet-5 for MNIST

Layer	Con guration
1st Convolutional	lters= 32, kernal size=5 5, stride= 1, activation= ReLu
1st Max Pooling	kernal size=2 2, stride= 2
2nd Convolutional	lters= 64, kernel size=5 5, stride= 1, activation= ReLu
2nd Max Pooling	kernal size=2 2, stride= 2
Fully Connected	lters= 1024, activation= ReLu
Output	lters= 10

C.1.4 VGG-CIFAR10

This model is based on VGG-16 [113] with adaptation to CIFAR-10 dataset based on [114]. It consists of 5 groups of convolution layers and 1 group of fully-connected layers with a total of 13 convolution layers and 2 fully-connected layers as shown in Table C.4. We use the same dropout con guration as [114]. Besides, stochastic gradient descent (SGD) is used to optimize the model with an initial learning rate of 0:001. We will stop updating the model once accuracy and attack success rate on validation set become steady (about 500 batches).

Table C.4: Main Architecture of VGG-CIFAR10

Layer	Con guration
2 Convolutional	lters= 64, kernal size=3 3, stride= 1, activation= ReLu
Max Pooling	kernal size=2 2, stride= 2
2 Convolutional	lters= 128, kernel size=3 3, stride= 1, activation= ReLu
Max Pooling	kernal size=2 2, stride= 2
3 Convolutional	lters= 256, kernal size=3 3, stride= 1, activation= ReLu
Max Pooling	kernal size=2 2, stride= 2
3 Convolutional	lters= 512, kernal size=3 3, stride= 1, activation= ReLu
Max Pooling	kernal size=2 2, stride= 2
3 Convolutional	lters= 512, kernal size=3 3, stride= 1, activation= ReLu
Max Pooling	kernal size=2 2, stride= 2
Fully Connected	lters= 512, activation= ReLu
Fully Connected	lters= 10, activation= softmax

C.2 Generating Adaptive Perturbation mask For Direct Targeted Misclassi cation

As explained in Section 5.4.2, the problem we tackle in this work is substantially di erent and di cult than that studied by [93] in the sense that we focus on targeted misclassi cation by means of a small perturbation and its magnitude must be well constrained for the sake of stealthiness. One may wonder why not solve the optimization problem 5.3 directly to achieve a targeted misclassi cation but instead generate an adaptive perturbation for poisoned training. Here, we demonstrate that stealthiness cannot be guaranteed if targeted misclassi cation is enabled directly by the generated adaptive perturbation mask. Speci cally, we choose 5 pairs of targets (c, t). For each of them, we vary the magnitude constraint from 10 to 40 to see if applying the generated adaptive perturbation mask to instances from class c can cause misclassi cation as target class t directly. The instances are drawn from the testing set and we measure the corresponding attack success rate . The results are averaged among the 5 pairs and reported in Table C.5.

Table C.5: Attack Success Rate w.r.t. Various Magnitude Constraint on Adaptive Perturbation Mask

Magnitude Constraint	10	20	30	40
Attack Success Rate	1.6%	28.38%	74.68%	90.76%

Accordingly, an adaptive perturbation mask with max intensity of 10 can rarely succeed to cause targeted misclassification. In particular, to obtain an attack success rate above 90%, we have to apply a perturbation mask with max intensity as large as 40, which would inevitably fail to achieve stealthiness considered as one of the most important design goals.

Bibliography

[1] Gu, T. , B. Dolan-Gavitt , and S. Garg (2017) Badnets: Identifying vulnerabilities in the machine learning model supply chain,arXiv preprint arXiv:1708.06733

[2] Chen, X. , C. Liu , B. Li , K. Lu , and D. Song (2017) Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning,arXiv preprint arXiv:1712.05526

[3] Liu, Y. , S. Ma , Y. Aafer , W.-C. Lee , J. Zhai , W. Wang , and X. Zhang (2018) Trojaning Attack on Neural Networks, in 25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018,The Internet Society.

[4] Chen, M. , S. Mao , and Y. Liu (2014) Big data: a survey, Mobile Networks and Applications, 19(2), pp. 171 209.

[5] Manyika, J. , M. Chui , B. Brown , J. Bughin , R. Dobbs , C. Roxburgh , and A. H. Byers (2011) Big data: The next frontier for innovation, competition, and productivity, .

[6] Mell, P. and T. Grance (2009) The NIST de nition of cloud computing, National Institute of Standards and Technology, 53(6), p. 50.

[7] Jansen, W. A. (2011) Cloud hooks: Security and privacy issues in cloud computing, in System Sciences (HICSS), 2011 44th Hawaii International Conference on, IEEE, pp. 1 10.

[8] Zhou, M. , R. Zhang , W. Xie , W. Qian , and A. Zhou (2010) Security and privacy in cloud computing: A survey, in Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on, IEEE, pp. 105 112.

[9] Takabi, H. , J. B. Joshi , and G.-J. Ahn (2010) Security and Privacy Challenges in Cloud Computing Environments.IEEE Security & Privacy, 8(6), pp. 24 31.

[10] Dyer, J. and N. Zhang (2013) Security issues relating to inadequate authentication in MapReduce applications, in High Performance Computing and Simulation (HPCS), 2013 International Conference on, IEEE, pp. 281 288.

[11] Wei, W. , J. Du , T. Yu , and X. Gu (2009) Securemr: A service integrity assurance framework for mapreduce, in Computer Security Applications Conference, 2009. ACSAC'09. Annual, IEEE, pp. 73 82.

[12] Xiao, Z. and Y. Xiao (2011) Accountable MapReduce in cloud computing, in Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS 11), pp. 1082 1087.

[13] Wang, Y. and J. Wei (2011) Viaf: Veri cation-based integrity assurance framework for mapreduce, in Cloud Computing (CLOUD), 2011 IEEE International Conference on, IEEE, pp. 300 307.

[14] Huang, C. , S. Zhu , and D. Wu (2012) Towards trusted services: Result veri cation schemes for MapReduce, in Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, IEEE, pp. 41 48.

[15] Yoon, E. and A. Sqcuicciarini (2014) Toward Detecting Compromised MapReduce Workers through Log Analysis, in Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, pp. 41 50.

[16] Paladi, N. and A. Michalas (2014) â€œOne of our hosts in another countryâ€•: Challenges of data geolocation in cloud storage, in Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE), 2014 4th International Conference on, IEEE, pp. 1 6.

[17] Ateniese, G. , R. Di Pietro , L. V. Mancini , and G. Tsudik (2008) Scalable and e cient provable data possession, in Proceedings of the 4th international conference on Security and privacy in communication netowrks, ACM, p. 9.

[18] Ateniese, G. , R. Burns , R. Curtmola , J. Herring , O. Khan , L. Kissner , Z. Peterson , and D. Song (2011) Remote data checking using provable data possession, ACM Transactions on Information and System Security (TISSEC), 14(1), p. 12.

[19] Juels, A. and B. S. Kaliski Jr (2007) PORs: Proofs of retrievability for large les, in Proceedings of the 14th ACM conference on Computer and communications security Acm, pp. 584 597.

[20] Shacham, H. and B. Waters (2008) Compact proofs of retrievability, in Advances in Cryptology-ASIACRYPT 2008, Springer, pp. 90 107.

[21] Bowers, K. D., A. Juels, and A. Oprea (2009) HAIL: a high-availability and integrity layer for cloud storage, in Proceedings of the 16th ACM conference on Computer and communications security, ACM, pp. 187 198.

[22] Curtmola, R., O. Khan, R. Burns, and G. Ateniese (2008) MR-PDP: Multiple-replica provable data possession, in Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on, IEEE, pp. 411 420.

[23] Barsoum, A. F. and M. A. Hasan (2011) On Verifying Dynamic Multiple Data Copies over Cloud Servers, IACR Cryptology ePrint Archive, 2011, p. 447.

[24] Barga, R. and V. Fontama Predictive analytics with Microsoft Azure machine learning, Springer.

[25] Google, TensorFlow, https://www.tensorflow.org/.

[26] Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012) Imagenet classification with deep convolutional neural networks, in Advances in neural information processing systems, pp. 1097 1105.

[27] McCann, B., J. Bradbury, C. Xiong, and R. Socher (2017) Learned in translation: Contextualized word vectors, in Advances in Neural Information Processing Systems, pp. 6297 6308.

[28] Abdel-Hamid, O., A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu (2014) Convolutional neural networks for speech recognition, IEEE/ACM Transactions on audio, speech, and language processing, 22(10), pp. 1533 1545.

[29] Huang, W. and J. W. Stokes (2016) MtNet: a multi-task neural network for dynamic malware classification, in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, pp. 399 418.

[30] Ruan, G. and Y. Tan (2010) A three-layer back-propagation neural network for spam detection using artificial immune concentration, Soft computing, 14(2), pp. 139 150.

[31] Taigman, Y., M. Yang, M. Ranzato, and L. Wolf (2014) Deepface: Closing the gap to human-level performance in face verification, in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1701 1708.

[32] Bojarski, M. , D. Del Testa , D. Dworakowski , B. Firner , B. Flepp , P. Goyal , L. D. Jackel , M. Monfort , U. Muller , J. Zhang , et al. (2016) End to end learning for self-driving cars, arXiv preprint arXiv:1604.07316

[33] Biggio, B. , I. Corona , D. Maiorca , B. Nelson , N. 'rndi¢ , P. Laskov , G. Giacinto , and F. Roli (2013) Evasion attacks against machine learning at test time, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, pp. 387 402.

[34] Biggio, B. , B. Nelson , and P. Laskov (2012) Poisoning attacks against support vector machines, arXiv preprint arXiv:1206.6389.

[35] Papernot, N. , P. McDaniel , I. Goodfellow , S. Jha , Z. B. Celik , and A. Swami (2017) Practical black-box attacks against machine learning, in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ACM, pp. 506 519.

[36] Zhang, O. Q. , M. Kirchberg , R. K. Ko , and B. S. Lee (2011) How to track your data: The case for cloud computing provenance, in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, IEEE, pp. 446 453.

[37] Ikeda, R. , H. Park , and J. Widom Provenance for Generalized Map and Reduce Work ows, in CIDR 2011, Stanford InfoLab.
URL http://ilpubs.stanford.edu:8090/985/

[38] Simmhan, Y. L. , B. Plale , and D. Gannon (2005) A survey of data provenance in e-science, ACM Sigmod Record, 34(3), pp. 31 36.

[39] Freire, J. , D. Koop , E. Santos , and C. T. Silva (2008) Provenance for computational tasks: A survey, Computing in Science & Engineering 10(3), pp. 11 21.

[40] Muniswamy-Reddy, K.-K. , D. A. Holland , U. Braun , and M. I. Seltzer (2006) Provenance-Aware Storage Systems. in USENIX Annual Technical Conference, General Track, pp. 43 56.

[41] Muniswamy-Reddy, K.-K. and M. Seltzer (2010) Provenance as rst class cloud data, ACM SIGOPS Operating Systems Review, 43(4), pp. 11 16.

[42] Akoush, S. , R. Sohan , and A. Hopper (2013) HadoopProv: Towards Provenance As a First Class Citizen in MapReduce, in Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance, TaPP '13, USENIX Association, Berkeley, CA, USA, pp. 11:1 11:4.
URL http://dl.acm.org/citation.cfm?id=2482949.2482963

[43] The Kepler Project, http://kepler-project.org .

[44] The Taverna Project, http://taverna.sourceforge.net.

[45] Muniswamy-Reddy, K.-K. , P. Macko , and M. I. Seltzer (2010) Provenance for the Cloud. in FAST, vol. 10, pp. 15 14.

[46] Park, H. , R. Ikeda , and J. Widom (2011) RAMP: A System for Capturing and Tracing Provenance in MapReduce Work ows, in 37th International Conference on Very Large Data Bases (VLDB) Stanford InfoLab.

[47] Hasan, R. , R. Sion , and M. Winslett (2007) Introducing secure provenance: problems and challenges, in Proceedings of the 2007 ACM workshop on Storage security and survivability ACM, pp. 13 18.

[48] McDaniel, P. , K. R. Butler , S. E. McLaughlin , R. Sion , E. Zadok , and M. Winslett (2010) Towards a Secure and E cient System for End-to-End Provenance. in TaPP.

[49] Hasan, R. , R. Sion , and M. Winslett (2009) The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance. in FAST, vol. 9, pp. 1 14.

[50] Bates, A. , B. Mood , M. Valafar , and K. Butler (2013) Towards secure provenance-based access control in cloud environments, in Proceedings of the third ACM conference on Data and application security and privacy ACM, pp. 277 284.

[51] Braun, U. , A. Shinnar , and M. I. Seltzer (2008) Securing Provenance. in HotSec

[52] Rosenthal, A. , L. Seligman , A. Chapman , and B. T. Blaustein (2009) Scalable Access Controls for Lineage. in Workshop on the Theory and Practice of Provenance

[53] Ni, Q. , S. Xu , E. Bertino , R. Sandhu , and W. Han (2009) An access control language for a general provenance model, in Secure Data Management Springer, pp. 68 88.

[54] Roy, I. , S. T. Setty , A. Kilzer , V. Shmatikov , and E. Witchel (2010) Airavat: Security and Privacy for MapReduce. in NSDI, vol. 10, pp. 297 312.

[55] Hadoop Apache, http://hadoop.apache.org .

[56] Dean, J. and S. Ghemawat (2008) MapReduce: simpli ed data processing on large clusters, Communications of the ACM, 51(1), pp. 107 113.

[57] Vavilapalli, V. K. , A. C. Murthy , C. Douglas , S. Agarwal , M. Konar , R. Evans , T. Graves , J. Lowe , H. Shah , S. Seth , et al. (2013) Apache hadoop yarn: Yet another resource negotiator, in Proceedings of the 4th annual Symposium on Cloud Computing, ACM, p. 5.

[58] Tan, J. , X. Pan , E. Marinelli , S. Kavulya , R. Gandhi , and P. Narasimhan (2010) Kahuna: Problem diagnosis for mapreduce-based cloud computing environments, in Network Operations and Management Symposium (NOMS), 2010 IEEE, IEEE, pp. 112 119.

[59] AspectJ, http://eclipse.org/aspectj/ .

[60] Javassist, http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/ .

[61] HDFS Architecture, http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html .

[62] Geist, M. , Location matters up in the cloud, http://www.thestar.com/business/2010/12/04/geist_location_matters_up_in_the_cloud.html .

[63] Peterson, Z. N. , M. Gondree , and R. Beverly (2011) A position paper on data sovereignty: the importance of geolocating data in the cloud, in Proceedings of the 8th USENIX conference on Networked systems design and implementation.

[64] Benson, K. , R. Dowsley , and H. Shacham (2011) Do you know where your cloud les are? in Proceedings of the 3rd ACM workshop on Cloud computing security workshop, ACM, pp. 73 82.

[65] Gondree, M. and Z. N. Peterson (2013) Geolocation of data in the cloud, in Proceedings of the third ACM conference on Data and application security and privacy, ACM, pp. 25 36.

[66] Watson, G. J. , R. Safavi-Naini , M. Alimomeni , M. E. Locasto , and S. Narayan (2012) LoSt: location based storage, in Proceedings of the 2012 ACM Workshop on Cloud computing security workshop, ACM, pp. 59 70.

[67] Microsoft , Azure Machine Learning Studio, https://studio.azureml.net/ .

[68] Fredrikson, M. , S. Jha , and T. Ristenpart (2015) Model inversion attacks that exploit con dence information and basic countermeasures, in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, pp. 1322 1333.

[69] Fredrikson, M. , E. Lantz , S. Jha , S. Lin , D. Page , and T. Ristenpart Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing. .

[70] Tramèr, F. , F. Zhang , A. Juels , M. K. Reiter , and T. Ristenpart (2016) Stealing Machine Learning Models via Prediction APIs. in USENIX Security Symposium, pp. 601 618.

[71] Cortes, C. and V. Vapnik (1995) Support-vector networks, Machine learning, 20(3), pp. 273 297.

[72] Drucker, H. , D. Wu , and V. N. Vapnik (1999) Support vector machines for spam categorization, IEEE Transactions on Neural networks, 10(5), pp. 1048 1054.

[73] Fierrez-Aguilar, J. , J. Ortega-Garcia , J. Gonzalez-Rodriguez , and J. Bigun (2005) Discriminative multimodal biometric authentication based on quality measures, Pattern recognition, 38(5), pp. 777 779.

[74] Mukkamala, S. , G. Janoski , and A. Sung (2002) Intrusion detection using neural networks and support vector machines, in Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on, vol. 2, IEEE, pp. 1702 1707.

[75] Papernot, N. , P. McDaniel , S. Jha , M. Fredrikson , Z. B. Celik , and A. Swami (2016) The limitations of deep learning in adversarial settings, in Security and Privacy (EuroS&P), 2016 IEEE European Symposium on, IEEE, pp. 372 387.

[76] McDaniel, P. , N. Papernot , and Z. B. Celik (2016) Machine learning in adversarial settings, IEEE Security & Privacy, 14(3), pp. 68 72.

[77] Moosavi Dezfooli, S. M. , A. Fawzi , and P. Frossard (2016) Deepfool: a simple and accurate method to fool deep neural networks, in Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), EPFL-CONF-218057.

[78] Jang, U. , X. Wu , and S. Jha (2017) Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning, in Proceedings of the 33rd Annual Computer Security Applications Conference, ACM, pp. 262 277.

[79] Mozaffari-Kermani, M. , S. Sur-Kolay , A. Raghunathan , and N. K. Jha (2015) Systematic poisoning attacks on and defenses for machine learning in healthcare, IEEE journal of biomedical and health informatics, 19(6), pp. 1893 1905.

[80] Shen, S., S. Tople , and P. Saxena (2016) A uror: defending against poisoning attacks in collaborative deep learning systems, inProceedings of the 32nd Annual Conference on Computer Security Applications, ACM, pp. 508 519.

[81] Bottou, L. (2010) Large-scale machine learning with stochastic gradient descent, in Proceedings of COMPSTAT'2010, Springer, pp. 177 186.

[82] Rumelhart, D. E. , G. E. Hinton , R. J. Williams , et al. Learning representations by back-propagating errors,Cognitive modeling, 5(3), p. 1.

[83] Shin, H.-C. , H. R. Roth , M. Gao , L. Lu , Z. Xu , I. Nogues , J. Yao , D. Mollura , and R. M. Summers (2016) Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning, IEEE transactions on medical imaging, 35(5), pp. 1285 1298.

[84] Metsis, V. , I. Androutsopoulos , and G. Paliouras (2006) Spam ltering with naive bayes-which naive bayes? inCEAS, vol. 17, pp. 28 69.

[85] LeCun, Y. , C. Cortes , and C. J. Burges (2010) MNIST handwritten digit database, AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, 2.

[86] Klimt, B. and Y. Yang (2004) Introducing the Enron Corpus. in CEAS.

[87] Abadi, M. , P. Barham , J. Chen , Z. Chen , A. Davis , J. Dean , M. Devin , S. Ghemawat , G. Irving , M. Isard , et al. (2016) TensorFlow: A System for Large-Scale Machine Learning. inOSDI, vol. 16, pp. 265 283.

[88] Cao, X. and N. Z. Gong (2017) Mitigating evasion attacks to deep neural networks via region-based classi cation, inProceedings of the 33rd Annual Computer Security Applications Conference, ACM, pp. 278 287.

[89] Grosse, K. , P. Manoharan , N. Papernot , M. Backes , and P. McDaniel (2017) On the (statistical) detection of adversarial examples,arXiv preprint arXiv:1702.06280

[90] Huang, L. , A. D. Joseph , B. Nelson , B. I. Rubinstein , and J. Tygar (2011) Adversarial machine learning, in Proceedings of the 4th ACM workshop on Security and arti cial intelligence, ACM, pp. 43 58.

[91] Muñoz-González, L. , B. Biggio , A. Demontis , A. Paudice , V. Wongrassamee , E. C. Lupu , and F. Roli (2017) Towards poisoning of deep learning algorithms with back-gradient optimization, in Proceedings of the 10th ACM Workshop on Arti cial Intelligence and Security, ACM, pp. 27 38.

[92] Ji, Y., X. Zhang, and T. Wang (2017) Backdoor attacks against learning systems, in CNS, 2017 IEEE Conference on, IEEE, pp. 1–9.

[93] Moosavi-Dezfooli, S., A. Fawzi, O. Fawzi, and P. Frossard (2017) Universal Adversarial Perturbations, in 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, pp. 86–94.

[94] Narayan, K. (2018), Insider Threat in the Cloud, https://www.skyhighnetworks.com/cloud-security-blog/5-devious-instances-insider-threat-cloud/.

[95] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009) Imagenet: A large-scale hierarchical image database, in Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE, pp. 248–255.

[96] Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick (2014) Microsoft coco: Common objects in context, in European conference on computer vision, Springer, pp. 740–755.

[97] Krasin, I. and T. Duerig (2016), Google Open Images Dataset, https://research.googleblog.com/2016/09/introducing-open-images-dataset.html.

[98] Le, Q. V., M. Ranzato, R. Monga, M. Devin, G. Corrado, K. Chen, J. Dean, and A. Y. Ng (2012) Building high-level features using large scale unsupervised learning, in Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012

[99] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. (2015) Going deeper with convolutions, CVPR.

[100] Bengio, Y. (2012) Deep learning of representations for unsupervised and transfer learning, in Proceedings of ICML Workshop on Unsupervised and Transfer Learning, pp. 17–36.

[101] Cotter, A., O. Shamir, N. Srebro, and K. Sridharan (2011) Better mini-batch algorithms via accelerated gradient methods, in Advances in neural information processing systems, pp. 1647–1655.

[102] Bottou, L. (2012) Stochastic gradient descent tricks, in Neural networks: Tricks of the trade, Springer, pp. 421–436.

[103] LeCun, Y. , L. Bottou , G. B. Orr , and K.-R. Müller (1998) E cient backprop, in Neural networks: Tricks of the trade, Springer, pp. 9 50.

[104] Goodfellow, I. , Y. Bengio , and A. Courville (2016) Deep Learning MIT Press, http://www.deeplearningbook.org .

[105] LeCun, Y. , L. Bottou , Y. Bengio , and P. Haffner (1998) Gradient-based learning applied to document recognition, Proceedings of the IEEE, 86(11), pp. 2278 2324.

[106] Krizhevsky, A. (2009) Learning multiple layers of features from tiny images, .

[107] Sermanet, P. and Y. LeCun (2011) Tra c sign recognition with multi-scale convolutional networks, in Neural Networks (IJCNN), The 2011 International Joint Conference on, IEEE, pp. 2809 2813.

[108] Kingma, D. P. and J. Ba (2014) Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[109] French, R. M. (1999) Catastrophic forgetting in connectionist networks, Trends in cognitive sciences, 3(4), pp. 128 135.

[110] McCloskey, M. and N. J. Cohen (1989) Catastrophic interference in connectionist networks: The sequential learning problem, in Psychology of learning and motivation, vol. 24, Elsevier, pp. 109 165.

[111] Klinger, E. and D. Starkweather (2008), pHash, http://www.phash.org/ .

[112] Google (2018), A Guide to TF Layers: Building a Convolutional Neural Network, https://www.tensorflow.org/tutorials/layers .

[113] Simonyan, K. and A. Zisserman (2014) Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[114] Liu, S. and W. Deng (2015) Very deep convolutional neural network based image classi cation using small training sample size, in Pattern Recognition (ACPR), 2015 3rd IAPR Asian Conference on, IEEE, pp. 730 734.

[115] Bradski, G. (2000) The OpenCV Library, Dr. Dobb's Journal of Software Tools.

[116] Szegedy, C. , W. Zaremba , I. Sutskever , J. Bruna , D. Erhan , I. Goodfellow , and R. Fergus (2013) Intriguing properties of neural networks, arXiv preprint arXiv:1312.6199.

[117] Goodfellow, I. J. , J. Shlens , and C. Szegedy (2014) Explaining and harnessing adversarial examples, arXiv preprint arXiv:1412.6572.

[118] Liu, Y. , Y. Xie , and A. Srivastava (2017) Neural trojans, in Computer Design (ICCD), 2017 IEEE International Conference on, IEEE, pp. 45 48.

[119] Johnson, N. F. and S. Jajodia (1998) Exploring steganography: Seeing the unseen, Computer, 31(2).

[120] Provos, N. and P. Honeyman (2003) Hide and seek: An introduction to steganography, IEEE security & privacy, 99(3), pp. 32 44.

[121] Cox, I. , M. Miller , J. Bloom , J. Fridrich , and T. Kalker (2007) Digital watermarking and steganography, Morgan Kaufmann.

[122] Cheddad, A. , J. Condell , K. Curran , and P. Mc Kevitt (2010) Digital image steganography: Survey and analysis of current methods, Signal processing, 90(3), pp. 727 752.

[123] Shih, F. Y. (2017) Digital watermarking and steganography: fundamentals and techniques, CRC press.

[124] Nielsen, M. A. (2015), Neural networks and deep learning, .

Vita

Cong Liao

Cong Liao completed his Ph.D. studies in the College of Information Sciences and Technology at Pennsylvania State University. He was advised by Dr. Anna Squicciarini. His research interests include cloud security and adversarial machine learning. He also received a B.E. degree in Mechanical Design, Manufacturing and Automation from University of Electronic Science and Technology of China in 2011, and a M.S. degree in Robotics from University of Pennsylvania in 2013.