FINE-GRAINED ANOMALY DETECTION FOR IN DEPTH DATA PROTECTION

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Shagufta Mehnaz

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Elisa Bertino, Chair

Department of Computer Science

Dr. Mikhail Atallah

Department of Computer Science

Dr. Ninghui Li

Department of Computer Science

Dr. Christopher W. Clifton

Department of Computer Science

Approved by:

Dr. Christopher W. Clifton

Head of the School Graduate Program

This thesis is dedicated to my wonderful parents, Mohammad Majaz Hussain and Nazmun Nahar, and to my loving husband, Syed Rafiul Hussain.

ACKNOWLEDGMENTS

I sincerely thank my Ph.D. advisor, advisory committee members, and collaborators for their insightful advising and guidance. I am also grateful to all the faculty and staff of Purdue Computer Science for their continuous support.

I am forever in debt to my parents who always encouraged me to pursue my dreams and to my amazing sisters Bushra Naz and Madiha Naz who took care of my parents in my absence. I am grateful to my lovely family, my aunt Kamrun Nahar, and my uncle Mohammad Monirul Islam for being there whenever I needed them.

No words can explain how thankful I am to my loving husband, Syed Rafiul Hussain, for his never-ending inspiration throughout my doctoral study. I want to express my deepest gratitude to him for believing in me, for being the greatest support at the toughest times, and for all his sacrifices. Looking forward to an adventurous journey together and accomplishing many more milestones.

Thanks to my teachers and mentors from all the schools I have been fortunate enough to attend including Motijheel Ideal School, Viqarunnisa Noon College, and Bangladesh University of Engineering and Technology (BUET).

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Mehnaz, Shagufta PhD, Purdue University, August 2020. Fine-Grained Anomaly Detection for in Depth Data Protection. Major Professor: Elisa Bertino.

Data represent a key resource for all organizations we may think of. Thus, it is not surprising that data are the main target of a large variety of attacks. Security vulnerabilities and phishing attacks make it possible for malicious software to steal business or privacy sensitive data and to undermine data availability such as in recent ransomware attacks. Apart from external malicious parties, insider attacks also pose serious threats to organizations with sensitive information, e.g., hospitals with patients' sensitive information. Access control mechanisms are not always able to prevent insiders from misusing or stealing data as they often have data access permissions. Therefore, comprehensive solutions for data protection require combining access control mechanisms and other security techniques, such as encryption, with techniques for detecting anomalies in data accesses. In this thesis, we develop fine-grained anomaly detection techniques for ensuring in depth protection of data from malicious software, specifically, ransomware, and from malicious insiders. While anomaly detection techniques are very useful, in many cases the data that is used for anomaly detection are very sensitive, e.g., health data being shared with untrusted service providers for anomaly detection. The owners of such data would not share their sensitive data in plaintext with an untrusted service provider and this predicament undoubtedly hinders the desire of these individuals/organizations to become more data-driven. In this thesis, we have also built a privacy-preserving framework for real-time anomaly detection.

1. INTRODUCTION

Data stored in a file system can be compromised in many ways, e.g., by *ransomware* that encrypts files to breach data availability [1–3], or by *employees with malicious intentions* inside an organization. While encryption techniques do not protect data from ransomware, access control mechanisms are also not always able to prevent insiders from misusing or stealing data [4]. Therefore, comprehensive solutions for data protection require combining access control mechanisms and other security techniques, such as encryption, with techniques for detecting anomalies in data accesses. Apart from detecting malicious data accesses, anomaly detection is also used as the underlying technique to support efficient and effective decision making in many safety critical application domains, such as home security, patient monitoring, detecting cyber attacks in nuclear power plants, etc. For instance, identifying an emergency situation of a patient when signals from her health monitoring devices seem anomalous require effective real-time anomaly detection. Internet giants such as Google, Microsoft, and specialized companies (e.g., Anomaly) are already offering such anomaly-detection-as-a-service [5–7] for real-time data or for predictive maintenance. Although there exists significant work in the database domain to process queries on encrypted databases [8–10], the users of such anomaly-detection-as-a-service have to share their data in plaintext with the service providers to ensure effective anomaly detection. Needless to say, these services do not satisfy the privacy requirements of many applications that run analytics on sensitive data. This privacy issue exists despite the presence of Trusted Execution Environment that is used to secure the execution of the code dealing with sensitive data to prevent an attacker from stealing sensitive data by exploiting the vulnerabilities in the rest of the system.

In this dissertation, we, therefore, address the following research questions: *(1) how can we design a robust technique to identify cryptographic ransomware in real-time so that the damage to the file system can be minimized? (2) how to develop a fine-grained anomaly de-*

tection technique for ensuring in-depth protection of data from malicious insiders? and (3) is it possible to build a privacy-preserving framework for effective and real-time anomaly detection using edge computing where the users only share their encrypted data with the third-party edge service providers?

1.1 A Real-time Detection System Against Cryptographic Ransomware

Ransomware is a class of malware that has recently become very popular among cybercriminals. The goal of these cybercriminals is to obtain financial gain by holding the users' files hostage- either by encrypting the files or by locking the users' computers. In this work, we focus on *crypto ransomware* which encrypts files and asks users for a ransom in exchange of decryption keys that can be used to recover the files encrypted by the attacker. Among the recent ransomware attacks, Petya [1] is the deadliest one; it affected several pharmaceutical companies, banks, at least one airport and one U.S. hospital. Another massive ransomware that hit nearly 100 countries around the world is WannaCry [2]. This attack targeted not only large institutions but also any individual who could be reached. Systems can be attacked by ransomware in different ways. Most commonly, an organized crime group sets up multiple seemingly legitimate domains that contain malware to be automatically downloaded and discreetly/silently installed. Whenever a website in any of these domains is visited, the malware is downloaded without the user's knowledge.

Our key observation with most prevalent ransomware families is that the file system access pattern of the ransomware spawned processes is anomalous when compared to normal user operations. These processes read a file, encrypt the file contents, and write the encrypted contents to the file (alternatively, delete the old file and create a new file with encrypted contents). In order to identify such anomalous file accesses, we propose a solution dubbed as `RWGUard` with a threefold monitoring mechanism - the first one is for monitoring the processes, the second one is for monitoring changes in the file system, and finally, the last one is for monitoring access requests to decoy (bait) files deployed in the system. Unlike generic malware, ransomware wreak havoc systems within minutes (or seconds).

Therefore, analyzing processes' file usage patterns and searching for ransomware-like behaviors result in delayed detections. To address this challenge, we strategically deploy a number of decoy files in the system. Since in the normal cases a decoy file should not be written, whenever a ransomware process writes to such a decoy file, our *decoy monitoring* technique identifies the ransomware process instantaneously. Though some research work [11, 12] recommends using decoy files for detecting ransomware, such previous work does not present any analysis on the effectiveness of these decoy files with any real system design. *To the best of our knowledge, ours is the first work to empirically analyze the effectiveness of decoy techniques against ransomware.* The *process monitor* checks the running processes' I/O Request Packets (IRPs), e.g., IRP write, IRP create, IRP open, etc. While some existing approaches [13, 14] are signature-based and look for specific I/O request patterns, we exploit the *rapid encryption property* of ransomware [15], use a number of IRP metrics for building baseline profile for each running process, and utilize these baseline profiles for performing process anomaly detection. The *file change monitor* checks all changes performed on the files (e.g., create, delete, and write operations) to determine anomalous file changes. From our experimental observations, we have found that monitoring only the process activities [13, 14] or only the file changes [13, 16] is not sufficient for effective detection and results in both high false positives and high false negatives (e.g., we observed that the Cryptolocker ransomware encrypts files very slowly which sometimes evades process monitoring). Therefore, in this work, we enhance these existing techniques and combine them with the decoy monitoring module in order to provide an effective solution for protection against ransomware.

1.2 A Fine-grained Approach for Anomaly Detection in File System Accesses with Enhanced Temporal User Profiles

In the second contribution presented in this thesis, we aim to defend against insider attacks. An example of such insider attacks is the breach [17] at Sony Pictures Entertainment where at least one of the six attackers was a former system administrator with

extensive technical background and knowledge of Sony's internal systems. As attempts to steal data by malicious or compromised insiders are often characterized by unusual access patterns [18, 19], anomaly detection for data accesses can be a useful technique that well complements other security techniques, such as authentication, access control, and encryption.

The development of access anomaly detection techniques for file systems is challenging. In contrast to database anomaly detection techniques where SQL queries provide a structure [20, 21], lack of semantic information about accesses in the case of file systems makes the task of anomaly detection complex. A large number of false positives, i.e., non-anomalous accesses classified as anomalies, may slow down the entire system and disrupt the users' normal activities. On the other hand, a large number of false negatives, i.e., undetected anomalous accesses undermine the security protection. This trade-off between performance and security is a major challenge [22] in detecting anomalies in file system accesses. Moreover, a file system anomaly detection technique must add minimal overhead to the data access times. Therefore, in this work, we propose an effective and practical approach, dubbed as *Ghostbuster*, to detect anomalous accesses to the file system by creating fine-grained user profiles. Our proposed approach comprises of two phases: the *Profile Creation (PC)* phase and the *Anomaly Detection (AD)* phase. In the first phase, we collect detailed information about the file accesses resulting from a user's normal file system activities and create the user profile by using a combination of techniques including data mining. The profiles are later used in the second phase to monitor the file system usage and to raise alerts upon identifying anomalous activities.

1.3 Data Privacy for Real-time Anomaly Detection

Real-time anomaly detection in streaming data helps us quickly detect and resolve unanticipated situations, e.g., anomalies in patients' health monitoring devices, home security systems, and industrial IoT sensors. While Google [5], Microsoft [6], and specialized companies (e.g., Anomaly.io [7]) are already offering such real-time anomaly detection

as a service, their approaches do not comply with privacy-sensitive applications, such as healthcare and finance. The owners of such data would not share their sensitive data in plaintext with an untrusted service provider and this predicament undoubtedly hinders the desire of these individuals/organizations to become more data-driven.

In general, when the data is encrypted before off-loading, the edge server is unable to process the encrypted data efficiently. Most approaches based on Secure Multi-party Computation (SMC) include complicated techniques, such as Yao's garbled circuits [23] and oblivious transfer, that are impractical for real-time analytics. Many existing solutions for outsourced privacy-preserving anomaly detection rely on trusted third parties or assume the presence of non-colluding third parties, thus introducing weak links in the security chain [24], whereas solutions that perturb data for privacy [25] are susceptible to data reconstruction [26]. Solutions assuming co-operative anomaly detection [27] or crowdsourcing [28] using differential privacy do not directly apply to our scenario where there is a single data owner willing to outsource the anomaly detection task while simultaneously preserving the privacy of the data. Moreover, solutions leveraging differential privacy would require injecting a significant amount of fake data which in turn would reduce the utility significantly.

Motivated by these simultaneous needs for real-time anomaly detection and data privacy, we built a framework to enable privacy-preserving real-time anomaly detection on sensitive, time series, streaming data. This privacy-preserving framework, in combination with a lightweight and optimized encryption scheme (used to encrypt the data before off-loading it to the third-party service provider), facilitates efficient anomaly detection on encrypted data. We evaluated the proposed solution for windowed Gaussian anomaly detector – the most widely used anomaly detection algorithm [29], in terms of privacy, accuracy, latency, and communication cost. The proposed solution can be easily adapted for more complicated anomaly detection scenarios, e.g., dynamic (i.e., time-evolving) graph anomaly detection algorithms that aim to identify anomalous patterns using a moving window analysis.

1.4 Thesis Statement

Data are the main target of a large variety of attacks, e.g., ransomware attacks that encrypt files to breach data availability, or malicious insider attacks that breach data confidentiality. Existing security techniques such as access control, encryption, and authentication need to be strengthened with anomaly detection techniques in order to achieve robust security while we also need to make sure that these anomaly detection techniques are accessible for applications that deal with privacy sensitive data.

2. RWGUARD: A REAL-TIME DETECTION SYSTEM AGAINST CRYPTOGRAPHIC RANSOMWARE

2.1 Introduction

Ransomware is a class of malware that has recently become very popular among cybercriminals. The goal of these cybercriminals is to obtain financial gain by holding the users' files hostage- either by encrypting the files or by locking the users' computers. In this chapter, we focus on *crypto ransomware* that asks users for a ransom in exchange of decryption keys that can be used to recover the files encrypted by the attacker. Such ransomware is now a significant threat to both individuals and organizations. Among recent ransomware attacks, Petya [1] is the deadliest one; it affected several pharmaceutical companies, banks, at least one airport and one U.S. hospital. Another massive ransomware that hit nearly 100 countries around the world is WannaCry [2]. This attack targeted not only large institutions but also any individual who could be reached. While ransomware has maintained prominence as one of the biggest threats since 2005, the first ransomware attack occurred in 1989 [30] and targeted the healthcare industry. The healthcare industry, which possesses very sensitive and critical information, still remains a top target.

Even though several techniques have been proposed for detecting malware, very few of them are specific to ransomware detection [13–16, 31–33]. Such existing techniques, however, have at least one of the following limitations: (a) impractically late detection when several files have already been encrypted [13, 16, 33], (b) failure to distinguish benign file changes from ransomware encryption [13–16, 31–33], (c) offline detection system that is unable to detect ransomware in real-time [13], (d) emphasis only on post-encryption phase which fails to recover files in most of the cases [32] or conflicts with secure deletion [15, 31], and (e) monitoring applications' actions only for a limited amount of time after their installation [33].

Problem and scope: In this work, we focus on the most critical requirement for a successful ransomware, i.e., *making the valuable resources (i.e., files, documents) unavailable to the user*, and design a solution, `RWGuard`, that protects against ransomware by detecting and stopping the ransomware processes at an early stage. Note that the ransomware families that lock the user's machine are out of the scope of this thesis.

Approach: `RWGuard` employs three monitoring techniques: decoy monitoring, process monitoring, and file change monitoring. Unlike generic malware, ransomware wreak havoc systems within minutes (or seconds). Therefore, analyzing processes' file usage patterns and searching for ransomware-like behaviors result in delayed detections. To address this challenge, we strategically deploy a number of decoy files in the system. Since in the normal cases a decoy file should not be written, whenever a ransomware process writes to such a decoy file, our *decoy monitoring* technique identifies the ransomware process instantaneously. Though some research work [11, 12] recommends using decoy files for detecting ransomware, such previous work does not present any analysis on the effectiveness of these decoy files with any real system design. *To the best of our knowledge, ours is the first work to empirically analyze the effectiveness of decoy techniques against ransomware.* The *process monitor* checks the running processes' I/O Request Packets (IRPs), e.g., IRP write, IRP create, IRP open, etc. While some existing approaches [13, 14] are signature-based and look for specific I/O request patterns, we exploit the *rapid encryption property* of ransomware [15], use a number of IRP metrics for building baseline profile for each running process, and utilize these baseline profiles for performing process anomaly detection. The *file change monitor* checks all changes performed on the files (e.g., create, delete, and write operations) to determine anomalous file changes. From our experimental observations, we have found that monitoring only the process activities [13, 14] or only the file changes [13, 16] is not sufficient for effective detection and results in both high false positives and high false negatives (e.g., we observed that the Cryptolocker ransomware encrypts files very slowly which sometimes evades process monitoring). *In this chapter, we enhance these existing techniques and combine them with the decoy monitoring module in order to provide an effective solution for protection against ransomware.*

If a potential encryption of a file (not a decoy) is identified, the next step is to determine whether the file is encrypted by a ransomware (referred to as *ransomware encryption*) or by a legitimate user (referred to as *benign encryption*). Therefore, we also design a file classification mechanism that depending on the properties of a file, classifies the encryption as benign or malicious. In order to learn the user's file encryption behavior, we leverage an existing encryption utility (that utilizes cryptographic library CryptoAPI, e.g., Kryptel [34]) to be used by end-users and applications. Finally, our approach includes a mechanism that places hooks and intercepts calls to the functions in CryptoAPI library so as to monitor all benign file encryption.

Contributions: To summarize, `RWGuard` makes the following contributions:

1. A decoy based ransomware detection technique that is able to identify ransomware processes in real-time.

2. A ransomware surveillance system that employs both *process* and *file change* monitoring (to detect ransomware encrypting files other than decoy).

3. A classification mechanism to distinguish benign file changes from ransomware encryption by hooking relevant CryptoAPI functions and learning the user's file encryption behaviors.

4. An extensive evaluation of our ransomware detection system on 14 most prevalent ransomware families to date.

2.2 Background

2.2.1 Hybrid Cryptosystem

A hybrid cryptosystem allows the ransomware to use different symmetric keys for encryption of different files while using a single asymmetric key pair. The attacker generates the asymmetric public-private key pair on its own command and control infrastructure. The ransomware code generates a unique symmetric key for each file to be encrypted and then

encrypts these symmetric keys with its public key. These encrypted symmetric keys are then left with the encrypted files. At this point, the user needs to pay the ransom to get the private key with which it can first retrieve the symmetric keys, and then decrypt the files.

2.2.2 IRPLogger

All the I/O requests by processes that are sent to device drivers are packaged in I/O request packets (IRPs). These requests are generated for any file system operation, e.g., open, close, write, read, etc. IRPLogger leverages a mini-filter driver [35] that intercepts the I/O requests. An example of IRPLogger entry is:

```
<Timestamp, PID, IRP/FastIO, Operation (READ/WRITE/OPEN/CLOSE/CREATE)>
```

2.2.3 CryptoAPI

CryptoAPI is a Microsoft Windows platform specific cryptographic application programming interface (API). This API, included with Windows operating systems, provides services to secure Windows-based applications using cryptography. It includes functionalities for encrypting (*CryptEncrypt*) and decrypting (*CryptDecrypt*) data, generating cryptographically secure pseudo-random numbers (*CryptGenRandom*), authentication using digital certificates, etc.

2.2.4 Microsoft Detours Library

Detours is a library for instrumenting arbitrary Win32 functions in Windows-compatible processors. It intercepts Win32 functions by re-writing the in-memory code for target functions. Detours preserves the un-instrumented target function (callable through a trampoline) as a subroutine for use by the instrumentation.

2.3 RWGuard Design

2.3.1 Threat Model

In our threat model, we consider an adversary that installs crypto-ransomware on victim machines through seemingly legitimate but malicious domains. We consider the operating system to be trusted. Ransomware generally targets and encrypts files that the user creates and cares about, and the user account already has all the privileges to access these files. However, though the assumption that ransomware executes only with user-level privileges seems reasonable (as otherwise, it may be able to defeat any existing in-host protection mechanisms, e.g., anti-malware solutions), this assumption does not apply to all the ransomware cases. We have observed some exceptions to this assumption where ransomware samples affect only a predefined list of system files and if not detected/terminated, gain root access, shut down the system, and at the next boot up, perform full disk encryption and ask for a ransom payment. Hence, we also include these ransomware samples in our threat model. Moreover, a malicious insider in an organization may gain the knowledge of decoy files and build a customized ransomware to sabotage the organization (installed as a logical bomb to detonate after the insider leaves the organization). A further discussion on how our `RWGuard` system handles such situations is given in Section 2.5. Also, note that, a malware changing file contents with non-random bits (i.e, not an encryption) and demanding ransom payment is out of the scope of this thesis. In order to maintain their credibility and ensure their continued stream of income, crypto-ransomware attackers generally encrypt the files and return the private key once the ransom is paid.

2.3.2 Overview

Figure 2.1 shows the placement and the design overview of `RWGuard`. Any I/O request to the file system generated by any user space process first needs to be scheduled by the I/O scheduler. We leverage IRPLogger to fetch these system-wide file system access requests and parse those with our IRPParser.

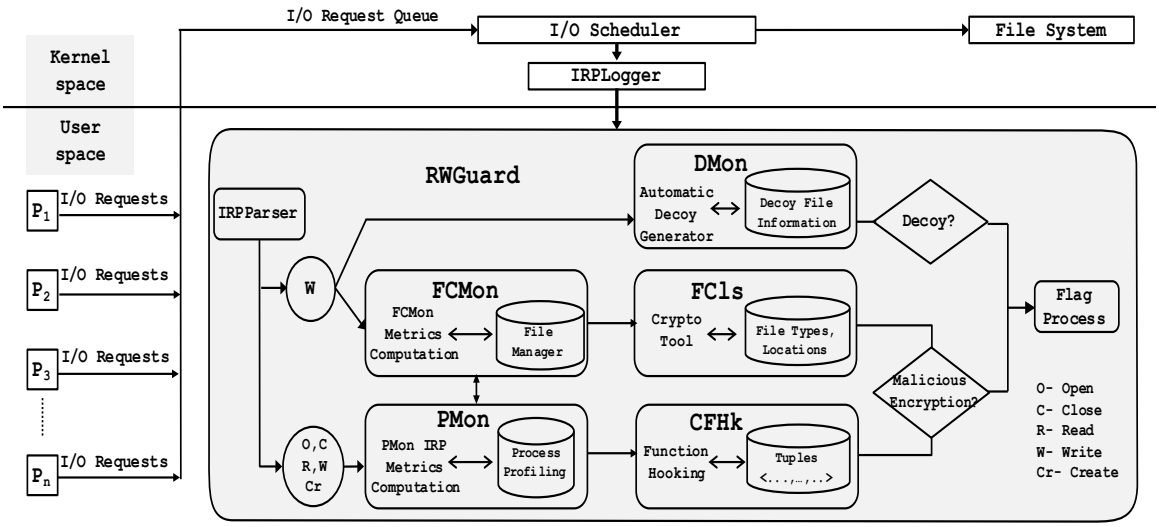


Figure 2.1.: Design overview of `RWGuard`

`RWGuard` consists of five modules: (1) *Decoy Monitoring (DMon)* module, (2) *Process Monitoring (PMon)* module, (3) *File Change Monitoring (FCMon)* module, (4) *File Classification (FCls)* module, and (5) *CryptoAPI Function Hooking (CFHk)* module. The *DMon* module considers only the IRP write requests as input and monitors whether there is any such request to a decoy file. The *PMon* and *FCMon* modules monitor process operations (IRP open, close, read, write, create) and file changes (IRP write), respectively. These two modules communicate in order to identify any process(es) making significant anomalous changes to the files. If such an event is identified, the *FCls* module checks the properties of the file and predicts the probability of the file change to be benign. Furthermore, the *CFHk* module checks whether a benign encryption (by the user) has been recorded for this file at the time of the file's significant change.

2.3.3 Decoy Monitoring (DMon) Module

The *DMon* module deploys decoy files that allow our system to identify a ransomware process in real-time. Since the decoy files should not be modified in normal situations, whenever a (ransomware) process tries to write such files, this module can immediately identify the process as malicious. Furthermore, the presence of a significant number of

decoy files (though of smaller sizes) increases the probability that a ransomware would encrypt one of these files even before trying to encrypt an original file. Hence, the advantage of using decoy files is twofold: (1) it allows the detection system to readily identify a malicious process, and (2) it delays the time when ransomware starts encrypting the original files and thus gives enough time for anomaly detection to complete its analysis and stop the malicious processes even before they start encrypting the original files (see Section 2.5.2 for the experimental data about the time required by `RWGuard` to complete the analysis). `RWGuard` decoy files are generated with an automated decoy generator tool that we discuss in details in Section 2.4.2. Note that, our decoy generator periodically modifies the decoy files so that even if a ransomware looks at the time when a file is last modified (to ensure that the file it encrypts is valuable to the user), it would not be able to recognize the decoy files.

2.3.4 Process Monitoring (PMon) Module

Unlike some existing approaches [13, 14] that look for *specific patterns* (e.g., read→ encrypt→delete) in the processes' I/O requests, we exploit the fact that ransomware typically attempts to encrypt data rapidly [15] (to maximize damage and minimize the chance of being detected) which leads to anomalous numbers of IRPs. Exploiting this property results in faster detection since IRPs can be logged well ahead of actual file operations. Our *PMon* module monitors the I/O requests made by the processes running on the system. Though IRP is the default mechanism for requesting I/O operations, many ransomware perform file operations using fast I/O requests. Fast I/O is specifically designed for rapid synchronous I/O operations on cached files, bypassing the file system and the storage driver stack. Therefore, in our design, we monitor both the IRPs and the fast I/O requests. A fast I/O read/write operation can be any of the types listed in Table 2.1. Given that ransomware processes encrypt files rapidly, the behavior of such processes has certain characteristics. Hence, in this module, we train a machine learning model that given a process's I/O requests, identifies the process as benign or ransomware. Ransomware that encrypt files

Table 2.1.: Fast I/O read and write types

READ types
FASTIO_READ
FASTIO_MDL_READ
FASTIO_READ_COMPRESSED
FASTIO_READ_COMPLETE_COMPRESSED

WRITE types
FASTIO_WRITE
FASTIO_MDL_WRITE
FASTIO_MDL_WRITE_COMPLETE
FASTIO_WRITE_COMPRESSED
FASTIO_MDL_WRITE_COMPLETE_COMPRESSED

Table 2.2.: Metrics for the *PMon* module

Metric #	Metric name
1	Number of IRP_WRITE requests
2	Number of FastIO_WRITE requests
3	Number of IRP_READ requests
4	Number of FastIO_READ requests
5	Number of IRP_OPEN requests
6	Number of FastIO_OPEN requests
7	Number of IRP_CREATE requests
8	Number of FastIO_CREATE requests
9	Number of IRP_CLOSE requests
10	Number of FastIO_CLOSE requests
11	Number of temporary file created

slowly may evade this module but are identified by the *FCMon* module as discussed in Section 2.3.5.

Process Profiling: In order to train the machine learning model, as a first step, we collect the IRPs (from this point, the term 'IRP' represents both I/O and fast I/O) of both benign and ransomware processes. Table 2.2 shows the IRP metrics used in this training phase which also includes the number of temporary files created by a process. The temporary files (.TMP) are usually created by ransomware to hold the data while copying or removing the original files. Once the profiles for benign and ransomware processes are built in the training phase, the *Process Profiling* component of the *PMon* module (Figure 2.1) stores the model parameters to check against the running processes' parameters in real-time (i.e., the test phase). The *PMon* module re-computes the metrics listed in Table 2.2 for each running process over a 3 seconds sliding window.

►**Training phase:** The data collection and classifier training steps are following:

1. **Data collection:** For the training set, we collect IRP data of processes from both ransomware samples and benign applications. We use nine of the most popular ransomware families, namely: Wannacry, Cerber, CryptoLocker, Petya, Mamba, TeslaCrypt, CryptoWall, Locky, and Jigsaw for the training phase. We also include benign processes, e.g.,

Table 2.3.: Performance evaluation for different machine learning techniques

Classifier	Accuracy (%)	ROC Area	True Positive Rate	False Positive Rate	Precision	Recall
Naive Bayes	80.07	0.69	0.80	0.70	0.75	0.80
Logistic Regression	81.22	0.72	0.81	0.66	0.77	0.81
Decision Tree	89.27	0.87	0.89	0.18	0.89	0.89
Random Forest	96.55	0.94	0.96	0.08	0.96	0.96

Explorer.exe, WmiPrvSE.exe, svchost.exe, FileSpy.exe, vmtoolsd.exe, csrss.exe, System, SearchFilterHost.exe, SearchProtocolHost.exe, SearchIndexer.exe, chrome.exe, GoogleUpdate.exe, services.exe, audiodg.exe, WinRAR.exe, taskhost.exe, drpbx.exe, lsass.exe, etc. It is important to note that most of the ransomware samples spawn multiple malicious processes during execution. Our final training dataset contains IRPs from 261 processes including both benign and malicious ones.

2. **Classifier training:** Using the training data, we train a machine learning classifier that, given a set of processes, is able to distinguish between ransomware and benign processes. In order to identify the best machine learning technique for this classification, we analyzed different classifiers, namely: *Naive Bayes* (using estimator classes), *Logistic Regression* (multinomial logistic regression model with a ridge estimator), *Decision Tree* [36], and *Random Forest* [37] classifiers. We used 10 fold cross validation on the obtained data set and measured accuracy, precision, recall, true positive rate and false positive rate for each of the above-mentioned classifiers. Table 2.3 presents a comparison of the classifiers used in our analysis. Figure 2.2 shows the results for all the classifiers in terms of ROC curves (which plot true positive rate against false positive rate). The low accuracy ($\sim$80%) of the naive Bayes classifier can be attributed to its class independence property. From our observation, ransomware usually employs a combination of read, write, open, and close requests which are correlated. Therefore, assuming that these parameters are independent of each other leads to a lower accuracy. The regression classifier works slightly better than the naive Bayes classifier with an accuracy of $\sim$81%. A logistic regression model searches for a single linear decision boundary in the feature space. Hence, the low accuracy can be attributed to the fact that our data does not have a linear boundary for decisions. The

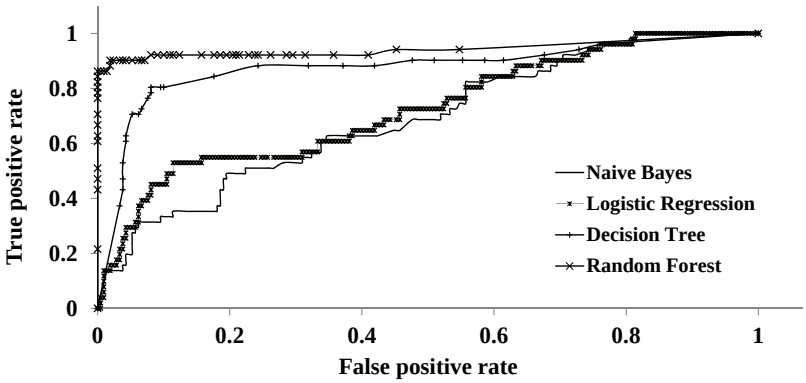


Figure 2.2.: ROC curves for different classifiers.

reason is that many ransomware make a large number of write/read requests as compared to the open/close requests. Therefore, the ideal decision boundary for our dataset would be non-linear.

The tree-based classifiers (random forest and decision tree) perform the best with accuracies of $\sim$97% and $\sim$89%, respectively. The reason is that the decision boundary for our data is non-linear and these classifiers build non-linear decision boundaries. However, the decision tree classifier is susceptible to over-fitting while random forest classifiers do not have this issue. Also, in terms of deployment, the random forest classifier is faster and more scalable compared to other classifiers. Therefore, finally, we use the random forest classifier in our `RWGuard` *PMon* module.

►**Test phase:** In the test phase, along with the nine families used for training, we add five more ransomware families in the experiment set: Vipasana, Satana, Radamant, Rex, and Matsnu. These samples are executed one at a time and depending on the spawned processes and their activities, the malicious processes are flagged. Details of the test phase results are given in Section 2.5.

File encryption: In our experiments, we observe that few benign processes, e.g., Chrome, VMware tools are sometimes classified as malicious by the machine learning model due to these processes' I/O request behaviors. Therefore, besides monitoring the process profiling metrics, it is important to monitor whether a particular process is responsible for any significant file changes. Hence, our *PMon* module considers *file encryption* as a significant

parameter (communicated by the *FCMon* module as described in Section 2.3.5) and identifies a process as malicious only if it encrypts files along with indications of anomalous I/O behaviors.

2.3.5 File Change Monitoring (FCMon) Module

This monitoring module can be configured to target a range of files from a single directory to the whole file system. It computes and stores the initial properties of the files (or, dynamically computes the properties when a file is created) and these properties are updated accordingly in the event of a file change. In real-time, the *FCMon* module looks for significant changes in those files after each write operation using the following metrics: (1) similarity, (2) entropy, (3) file type change, and (4) file size change. While some of these metrics have been used for ransomware detection in existing work [13, 16], our goal is to verify the fast detections by the *PMon* module and thereby minimize the false positive rates. In what follows, we describe the *File Manager* component of the *FCMon* module and present the details of the above metrics.

File Manager: This component stores the current properties of each file (e.g., file type, current entropy of a file, file size, last modified time etc.) so that any significant change in the files' properties can be detected upon a write operation. If a new file is created, this component computes the properties of the new file instantly and stores them in the map (map key: file name and path, key value: computed properties).

Metrics: The metrics of *FCMon* module are following:

1. *Similarity* metric: In comparison with a benign file change, e.g., modifying some of the existing text or adding some text, an encryption would result in data that is very dissimilar to the original data. Therefore, the similarity between a file's previous (before the write operation) and later (after the write operation) versions is an important factor to understand the characteristics of the file change. In order to compute the similarity between two versions, we use *sdhash*, a similarity-preserving hash function proposed by Roussev et al. [38] for generating the file hashes. The *sdhash* function outputs a score in

the range [0,100]. A score of 0 is obtained when we compute the similarity between two completely random arrays of data. Conversely, a score of 100 is obtained when we compute the similarity between two files that are exactly same. Hence, in the case of an encryption, this function outputs a value close to 0.

2. *Entropy* metric: Entropy, as it relates to digital information, is the measurement of randomness in a given set of values (data), i.e., when computed over a file, it provides information about the randomness of data in the file. Therefore, certainly, a user's data file in plaintext form has low entropy whereas its encrypted version would have a high entropy. Other than encrypted data, compressed data also has high entropy when compared to its plaintext form. A widely used entropy computation technique is Shannon entropy [39]. The Shannon entropy of an array of N bytes (assuming ASCII characters with values 0 to 255) can be computed as the following: $\sum_{i=0}^{255} P_i log_2 \frac{1}{P_i}$. Here, P_i is the probability that a randomly chosen byte from the array is i, (i.e., P_i =F_i/N) where F_i is the frequency of byte value i in the array. This equation returns a value in the range of [0,8]. For an absolutely even distribution of byte values in the array, the output value is 8. Since encrypted files have bytes more evenly distributed (when compared to its plaintext version), the Shannon entropy significantly increases after encryption and results in a value near 8.

3. *File type change* metric: A file generally does not change its type over the course of its existence. However, it is common for a number of ransomware families to change the file type after encryption. Therefore, whenever a file is written, we compare the file types before and after the write operation.

4. *File size change* metric: Unlike file type change, file size change is a common event, e.g., adding a large text to a document. However, this metric along with other metrics can determine if the file changes are benign or malicious.

Upon detecting a file write operation that results in a file type change or exceeds at least one of the given thresholds for the metrics, that is, similarity (score < 50), or entropy (value > 6), and/or significantly changes the file size, the *FCMon* module shares the recorded metrics with the *PMon*, *FCls*, and *CFHk* modules for further assessment.

2.3.6 File Classification (FCls) Module

After the *PMon* and *FCMon* modules collaboratively identify a process responsible for anomalous I/O behavior and file changes, our detection system classifies whether the file is encrypted by the ransomware or the change is due to a benign operation. Our *FCls* module performs this classification by learning the usage of the crypto-tool (a utility leveraging CryptoAPI used for user's sensitive files' encryption and decryption, e.g., Kryptel [34]) and profiling the user's encryption behavior. For example, if a file is encrypted which is from the same directory and has the same type of a previously benignly encrypted file, this module assigns a higher probability for this file to be benignly encrypted (however, a ransomware cannot abuse this idea as described in *CFHk* module in Section 2.3.7). If the probability for a file is too low to belong to the benignly encrypted class and if the file gets encrypted, a flag is raised immediately by the *FCls* module. In order to remove false negatives (i.e., ransomware encrypts a file which has a high probability of being benignly encrypted), the encryption information is validated with the *CFHk* module which intercepts benign encryptions.

Protecting sensitive files: If at the time of the ransomware attack the sensitive files are already in encrypted form, the ransomware could further encrypt those files which makes those files unavailable too. Note that the *FCMon* module may not be able to flag this event with high probability. The reason is that the entropy would not change significantly since both of the file versions (before and after the ransomware encryption) would have high entropy. To address such issue, we modify the permission settings for encrypted files, i.e., when a user encrypts a file using the crypto-tool, the only operations that we allow on that encrypted file are decryption and deletion (each of these operations requires the symmetric key used for encryption). Since it is impractical that someone would edit/modify an encrypted file before decryption, this permission setting suffices.

2.3.7 CryptoAPI Function Hooking (CFHk) Module

As described in Section 2.3.6, if the *FCls* module classifies the change to be the result of a possibly benign encryption, we need to further investigate whether the encryption was actually performed using the crypto-tool. Hence, the *CFHk* module places hooks at the beginning of the CryptoAPI library functions to redirect control of the execution to our custom-written functions. Figure 2.3 shows an example of hooking the 'CryptEncrypt' function included in the CryptoAPI library. Whenever a process calls the CryptEncrypt function to encrypt some file, the hook placed at the beginning of the CryptEncrypt function transfers control to a *shadow* CryptEncrypt function. This shadow CryptEncrypt function extracts a tuple $< key, algo, file, timestamp, process >$ for that particular call and stores this information in encrypted form for security purposes so that no other process can get access to this. The key for this encryption is derived from a secret password set by the user. Once the tuple is stored, the shadow CryptEncrypt function returns control to the original procedure, and the process continues its execution as if it had not been interrupted at all. The implementation details of this hooking procedure are discussed in Section 2.4.3.

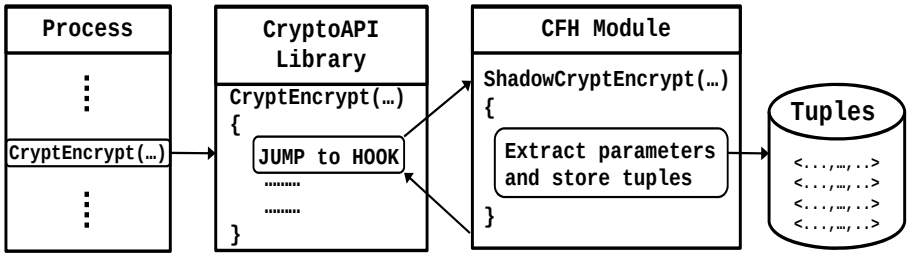


Figure 2.3.: CryptoAPI Function Hooking (CFHk) module

To identify whether a file encryption is performed using the crypto-tool, we simply search 'CryptEncrypt' tuples that are captured by the *CFHk* module.

- If such a tuple is not found, we terminate the process that resulted in the file change so that no further encryption can take place.

- If such a tuple is found, the encryption is either benign (no action required) or a ransomware using CryptoAPI is responsible for the encryption. In the second case, we can

recover all the files by using the key and $algo$ information from the tuples (details in Section 2.5.4). Since in our system we also store the $file$ information (by associating a *ReadFile* call with *CryptEncrypt*), we do not need to iterate over all the keys for a single file decryption which is an improvement over existing work [32].

Hence, the advantage of hooking the CryptoAPI library functions is twofold: (1) tracking all the benign encryption by the user, (2) recovering the ransomware encrypted files in the case that the ransomware dynamically links system-provided cryptographic libraries (i.e., Windows CryptoAPI).

2.4 `RWGuard` Implementation

2.4.1 IRPParser

While IRPLogger logs the I/O requests, the IRPParser component parses the log entries, extracts I/O requests, and provides these as input to the *DMon*, *PMon*, and *FCMon* modules accordingly.

2.4.2 Decoy File Generator

We have designed an automated decoy file generator tool that generates the decoy files based on the original file system and user preferences. By default, in each directory, it generates a decoy file with a name that is similar to one of the original files (selected at random or by the user depending on user preference) in that same directory so that the decoy files' names do not seem random to the ransomware. In order to make sure that the decoy files can be easily identified by the user, the naming options are selected based on the user's preferences which also makes the decoy files more unpredictable for the ransomware. The user is able to set different numbers of decoy files for different directories. In this way, the more sensitive files can be protected with a larger set of decoy files and also, manually setting the numbers makes it easier for the user to identify the decoy files during normal operations. The type extensions of the generated decoy files are: .txt, .doc, .pdf, .ppt, and

.xls whereas the contents of the files are generated from the contents of neighboring files. Although we did not observe *selective* behavior (e.g., checking file name, file content, etc. before encryption) in any of the ransomware we experimented with, our decoy design is resilient to such future advanced ransomware. Note that the sizes of the decoy files in our system are randomly taken from a range (typically from 1KB to few MBs) based on the sizes of the files in the original file system while the overall space overhead for decoy files is limited to 5% of the original file system size.

Table 2.4.: Hooked CryptoAPI functions

Function	Details
CryptEncrypt	Encrypts data
CryptGenKey	Generates a random cryptographic session key or a public/private key pair
CryptDeriveKey	Generates cryptographic session keys derived from a base data value
CryptExportKey	Exports a cryptographic key/key pair from a CSP
CryptGenRandom	Fills a buffer with cryptographically random bytes

2.4.3 CryptoAPI Function Hooking

In our *CFHk* module, we leverage the Detours library introduced in Section 2.2.4. Detours hooks a function by moving a specific number of bytes (generally five bytes) from the beginning of the original function's memory address to the newly created hook function. In this blank space of the original function, an unconditional JMP instruction is added that would transfer the control to the hook function. The hook function then performs the necessary operations (e.g., safely storing the keys and other parameters passed to the original function). At the end of these operations, another unconditional JMP instruction is added to transfer the control back to the original function. The compiled DLL file is placed into the registry key so that any process invoking the CryptoAPI functions would get hooked and our *CFHk* module would store information related to encryption. Table 2.4 lists the CryptoAPI functions we hook.

2.5 Evaluation

2.5.1 Experiment Dataset

While there exists different variants of ransomware, we build a comprehensive dataset from the most popular ransomware families: Locky, Cerber, Wannacry, Jigsaw, Cryptolocker, Mamba, Teslacrypt, Cryptowall, Petya, Vipasana, Satana, Radamant, Rex, and Matsnu. The ransomware samples are collected from VirusTotal [40], Open Malware [41], VXVault [42], Zelster [43], and Malc0de [44].

Note that among these samples, the first 9 families have been used in the training phase of the *PMon* module. However, we run each of these 14 ransomware samples (one at a time) in the detection phase to assess the detection effectivenesses and performance overheads of `RWGuard` modules. The reason behind not using the 5 samples for *PMon* module training is to measure how well this module performs with previously unseen ransomware samples.

2.5.2 Detection Effectiveness

We evaluate the performance of `RWGuard` by running the ransomware samples sequentially. Every time a ransomware sample is executed, we measure the time required for flagging each malicious process spawned by the ransomware. Once the ransomware is detected, we restore the system with a clean OS and execute the next ransomware sample.

Detection w/ Decoy Deployment:

We observe that ransomware detection with decoy deployment is extremely fast and ensures almost zero data loss. Note that the IRPParser component parses IRP logs collected in a 1 second cycle. Therefore, with the decoy deployment, our system can identify a ransomware process right in the next cycle of the process's decoy file write request.

Figure 2.4 shows the comparison between the cases of with and without decoy deployment in terms of the number of write, read, open, and close IRPs (along with the average values for all the ransomware) made by the ransomware samples until their detection (in

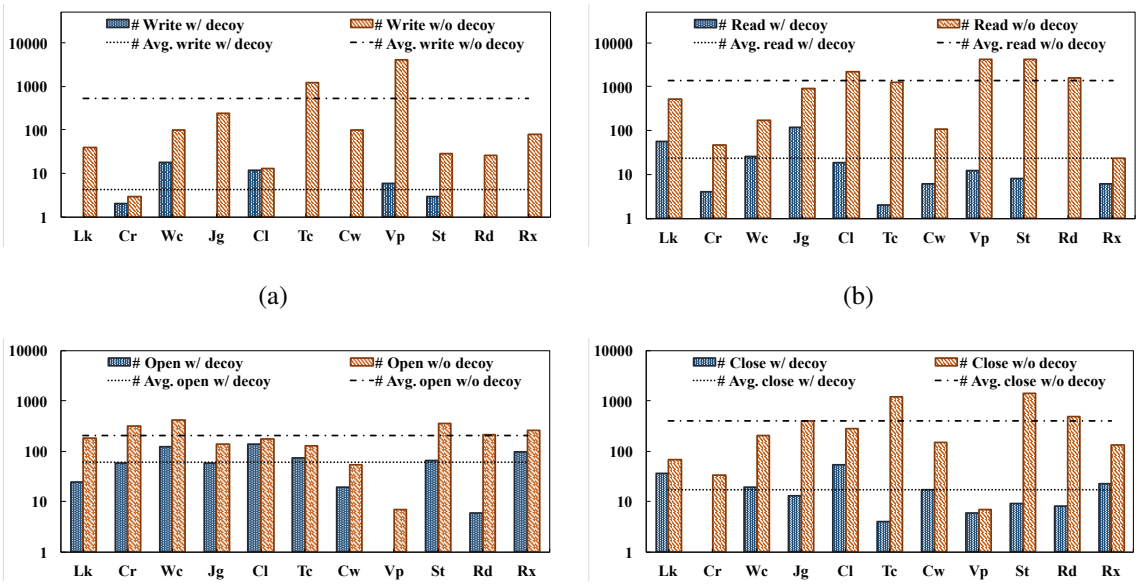


Figure 2.4.: Comparison between the cases of with and without decoy deployment in terms of the number of (a) *write*, (b) *read*, (c) *open*, and (d) *close* IRPs made by the ransomware samples until their detection (ransomware name abbreviations: Lk-Locky, Cr-Cerber, Wc-Wannacry, Jg-Jigsaw, Cl-Cryptolocker, Tc-Teslacrypt, Cw-Cryptowall, Vp-Vipasana, St-Satana, Rd-Radamant, Rx-Rex).

Figure 2.4(a), 2.4(b), 2.4(c), and 2.4(d), respectively). The number of IRPs (for each IRP type) for each ransomware family is computed by running the samples at least 5 times. We find that with decoy deployment, for each of these IRP types, there is an improvement of at least one order of magnitude. Hence, the ransomware processes could be identified as soon as they start making IRP requests, i.e., in real-time. *For ransomware Locky, Jigsaw, Teslacrypt, Cryptowall, Radamant, and Rex, we observe that the first IRP write requests they make are for decoy files (see Figure 2.4(a)) and thus are identified immediately. The Wannacry ransomware could make up to 18 IRP write requests (the highest) before it sends a write request for a decoy file (note that there can be multiple IRP write requests for a single file write operation).* An IRP write request is sent well ahead of the actual write operation and hence the actual number of files that can get encrypted before terminating the process is negligible (which also depends on the file size).

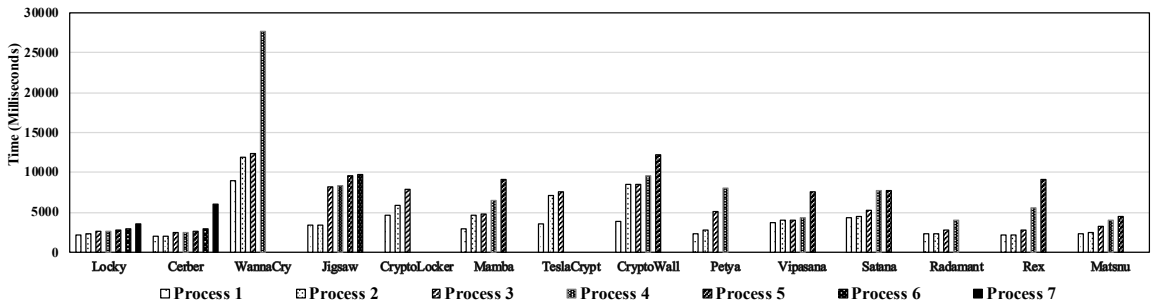


Figure 2.5.: Detection time required by `RWGuard` when there is no decoy deployment.

Note that, Figure 2.4 does not show comparisons for the following three ransomware families: Mamba, Petya, and Matsnu. In our experiments, we have found that samples from these families affect only a predefined list of system files (and if there is no detection system activated except the decoy monitoring, this is followed by gaining root access, shutting down the system, and at the next boot up, performing full disk encryption and asking for a ransom payment). As a result, our *DMon* module cannot identify such ransomware families (however, the *PMon* and *FCMon* modules can) and therefore, we omit the comparison for these three families in this section.

Detection w/o Decoy Deployment:

In order to further evaluate the effectiveness of `RWGuard`, we also consider an environment where there is no decoy file. This environment can be practical for the following two scenarios:

1. A ransomware encrypts only a predefined list of system files, i.e., even if the decoy files are deployed, the ransomware does not touch the decoy files (e.g., Mamba, Petya, and Matsnu ransomware families in our experiment dataset).

2. A malicious insider in an organization with the knowledge of decoy files' deployment can use customized ransomware to sabotage the organization and hold the ransomware responsible for this. Such an attack can be even launched as a logical bomb that can detonate after the insider has left the organization.

Figure 2.5 shows the time required to detect each of the samples (in milliseconds) while there is no decoy deployment in the system. The time computation starts when the ransomware sample is executed and ends when the corresponding process is flagged. Once the *PMon* and *FCMon* modules identify potential ransomware activity (i.e., malicious IRP/FastIO requests, significant file changes or encryption), the *FCls* and *CFHk* modules are communicated. If the file(s) that is (are) changed does (do) not belong to the 'benignly encrypted' class, and if there is (are) no corresponding encryption entry (entries) in the *CFHk* module, the process is immediately flagged. The average detection time for the first malicious processes spawned by all the ransomware is 3.45 seconds. However, we see that all the ransomware spawn multiple malicious processes which are detected at different times by our monitoring system. We observe that the average required time for detecting all the spawned processes is 8.87 seconds. As we can see from Figure 2.5, Locky, and Cerber spawn the highest number of malicious processes whereas CryptoLocker and TeslaCrypt spawn the lowest number of processes. According to our observation, most of the ransomware try to spawn processes with unique names or try to hide as system processes, e.g., explorer.exe. We also observe that different ransomware behave differently when the initially spawned malicious processes are killed by our system. For example, Wannacry sits idle for some time after the initial few processes are killed, before trying to spawn a new malicious process. This is the reason for the comparatively higher detection time for the last process in some of these ransomware.

Detection effectivenesses of different modules are discussed in the following:

- **Decoy Monitoring (DMon) module:** This module is the fastest to identify a ransomware process. Deploying a larger number of decoy files will result in even faster detection. For example, with a decoy generator that creates a shadow decoy file for each original file in the system, probabilistically, one out of each two write requests by a ransomware would belong to a decoy file.

- **Process Monitoring (PMon) and File Change Monitoring (FCMon) modules:** In most of the cases, the *PMon* module responded faster than the *FCMon* module in terms

of flagging a malicious process. Even before the ransomware starts performing encryption, the *PMon* module is able to identify the malicious activities by monitoring the IRPs. In contrast, the *FCMon* module responds only after a file has been changed significantly. However, we observe that few benign processes, e.g., Chrome, VMware tools are sometimes misclassified as malicious by the *PMon* module due to these processes' I/O request behaviors. Therefore, it is important to also consider the analysis by the *FCMon* module to better understand whether a particular process is responsible for any malicious file changes and to remove any false positives.

• **File Classification (FCls) and CryptoAPI Function Hooking (CFHk) modules:** After the *PMon* and *FCMon* modules' detection that a process is making significant changes in the file(s), the information of the file(s) are sent to the *FCls* module which then computes the probability of these changes being benign. The false negatives of this module correspond to the cases in which the ransomware encrypts a file which has a high probability of being encrypted by the user benignly. Such false negatives are, however, detected by the *CFHk* module which identifies if the file is actually encrypted using the provided crypto-tool. With a 100% accuracy, the *CFHk* module can identify whether an encryption is performed by a ransomware or is a benign encryption. This module never flags a benign encryption. The only case of false positives (negligible, $\sim$0.1%) we have observed in the *FCls* and *CFHk* modules is when the user performs *file compression* in a directory for the first time. However, a first time benign file encryption in a directory is not flagged as malicious since the *CFHk* module can intercept the benign encryption operations. Note that the *FCls* and *CFHk* modules do not flag any process unless that process is identified as suspicious by one of the monitoring modules.

2.5.3 Size of Encrypted Data

In terms of the number of files, samples from ransomware families Locky, Jigsaw, Teslacrypt, Cryptowall, Radamant, and Rex could not encrypt any file with decoy deployment. The malicious processes for these families are identified on their first IRP write request.

The numbers of IRP write requests made by ransomware families Cerber, Wannacry, Cryptolocker, Vipasana, and Satana before their detection are 2, 18, 12, 6, and 3, respectively, with decoy deployment. However, since an IRP write request is sent well ahead of the actual write operation and there can be multiple IRP write requests for a single file write, with decoy deployment, the average number of files lost is < 1 with only Wannacry and Cryptolocker being able to encrypt 1 file each before their malicious processes are killed. The average number of IRP requests made by the ransomware families without any decoy deployment is $\sim$538 (with the strong assumption that the ransomware can evade the decoy deployment which is not the case for most of the families) whereas the average number of files affected is < 10. Note that the number of files affected before detection depends not only on the number of IRP requests made but also on the time taken by a ransomware process to initiate the encryption routines (which is significant), type of encryption, size of the files, and the number of files the ransomware attempts to encrypt (this is because for each file the ransomware needs to generate a new key).

2.5.4 File Recovery

The *CFHk* module could recover all the files encrypted by the ransomware families: Locky, CryptoWall, and CryptoLocker. The encryption algorithms used by these samples are AES with CTR mode, AES in CBC mode, and AES, respectively. Note that the *CFHk* module in its current version cannot recover files that are encrypted using the ransomware's custom-written cryptographic library.

Table 2.5.: Memory overhead of `RWGuard`

Component	Memory consumed (KB)
Main Java module	14296
FCMon Entropy Calculator	7880
FCMon Similarity Index Calculator	5152
IRP Logger	42964

2.5.5 Performance Overhead

In the following, we discuss the performance overheads for different modules of `RWGuard`. The *DMon*, *FCls*, and *CFHk* modules have negligible overheads. The *DMon* module generates a single decoy file in each directory (if not set otherwise by the user) and randomly chooses the size of the decoy files from the range 1KB-5MBs while limiting the overall space overhead to 5% of the original file system size. At runtime, this module checks for decoy file write requests and modifies/regenerates the decoy files once per day at random times which has only a minimal overhead. The *FCls* module instantaneously classifies the files using file type and location information. The overhead for hooking a CryptoAPI function and computing and storing the corresponding tuple is a few milliseconds ($\leq$10ms) which is negligible and thus cannot interrupt a user's normal operations.

There is a main Java module which executes the IRPLogger, collects all the IRPs made in the system, parses the IRPs with IRPParser, and runs three parallel threads for *DMon*, *PMon*, and *FCMon* modules. The *FCMon* module consists of the components for computing the values of entropy and similarity index which use minimal CPU cycles since these are called only when there are write operations on the files. The memory usage of these components along with the main Java module is shown in Table 2.5. The average CPU usages for this main Java module and IRPLogger are 0.85% and 1.02%, respectively.

Overheads for different workloads. The performance overheads discussed above are recorded while running a web browser process and an integrated development environment (IDE) process along with regular operating system processes. However, in order to measure `RWGuard` detection performance and overheads for a heavy workload OS, we add several processes: two browsers (Chrome and Internet Explorer), two IDEs (Eclipse and PyChar), Windows Media Player, Skype, and other regular operating system processes. According to our experiments, this heavy workload does not significantly affect the time required by `RWGuard` for identifying ransomware processes while we have observed that IRPLogger and the Java module incur higher memory overhead (244456 KB and 45436 KB, respectively) due to this heavy workload. The detection time remaining unaffected by

the heavy workload can be attributed to the fact that RWGuard fetches IRPLogger entries every 2 seconds which does not depend on the number of entries logged (the number of log entries is much higher for the heavy workload case). Since parsing the IRP logs is not an expensive operation, for the heavy workload case, the detection time is not significantly changed. Also, the memory overheads for the *FCMon* metrics' calculation remain similar.

2.5.6 Comparison with Existing Approaches

Table 2.6 presents a comparison among RWGuard and other exiting ransomware detection techniques with respect to monitoring, detection, and recovery strategies.

Table 2.6.: Comparison of RWGuard with existing ransomware detection mechanisms

Solution	Real-time detection with decoy	Benign operation/ encryption profiling	File change monitoring	Process monitoring	Recovery of decryption key	Recovery of files
RWGuard	✓	✓	✓	✓	✓(partial)	✓(partial)
ShieldFS [31]	×	×	×	✓	×	✓
Unveil [13], CryptoDrop [16], Redemption [14]	×	×	✓	✓	×	×
PayBreak [32]	×	×	×	×	✓(partial)	✓(partial)
ElderRan [33]	×	×	×	✓	×	×
FlashGuard [15]	×	×	×	×	×	✓

2.6 Discussion and Limitations

Novelty: To the best of our knowledge, RWGuard with the decoy technique is the first system with very fast real-time (few milliseconds) detection capabilities. Even without the decoy deployment, the other monitoring modules are able to minimize the damage by identifying the ransomware processes at the time of their I/O requests. An average of 538 I/O write requests within the average detection time of 3.45 seconds shows how rapidly a ransomware attempts to encrypt the user's files while RWGuard exploits this property to terminate the ransomware at an early stage. Also, whereas the existing approaches are unable to distinguish benign file changes from malicious ones, the *FCls* module along with the *CFHk* module is able to overcome such false positives.

Inevitability: Our robust decoy design makes it impossible for the ransomware to recognize a decoy file by any of its properties. The ransomware would need to install some spyware and monitor the file activities in the system in order to determine which ones are modified by the end-users and applications and which are executed by our decoy tool. Moreover, obfuscation techniques can be used to make difficult for the ransomware to analyze the applications in order to determine which application is the decoy generator. Our integrated monitoring modules, *PMon* and *FCMon*, employ scrutiny on metrics that are inclusive of any malicious activity by the ransomware. For example, a smart ransomware that encrypts files slowly would still be detected by the *FCMon* module. While the monitoring modules *DMon*, *PMon*, and *FCMon* do not let a ransomware activity remain undetected (i.e., they prevent false negatives), the *FCls* and *CFHk* modules distinguish benign file operations from malicious ones (i.e., they prevent false positives). Hence, we argue that independently of the intelligence of modern ransomware, `RWGuard` raises the evasion bar for ransomware significantly.

File Recovery: Note that, the *CFHk* module monitors all (benign and ransomware) file encryption that leverage 'CryptoAPI' functions. Therefore, if a ransomware leveraging CryptoAPI library (3 of the 14 ransomware families that we have analyzed use this library) becomes successful in encrypting a set of files before our early detection, using the hooking mechanism, we can retrieve the parameters (including the decryption keys) of those specific cryptographic function calls and consequently restore the encrypted files. Our experiments (Section 2.5.4) show that the *CFHk* module is able to recover the files encrypted by the 3 ransomware families with a 100% success rate. The rest of the ransomware samples experimented in our evaluation did not use CryptoAPI but their custom-written cryptographic library. Moreover, code obfuscation is a common technique used by the modern ransomware families. Obfuscation strategies, such as incremental packing and unpacking, make it more difficult to identify cryptographic primitives in the ransomware binary. While there are techniques (e.g., [45]) that look for cryptographic operations in the process memory, we have not incorporated those in our system due to their huge performance overhead.

Limitations: While the *DMon* module is quick in identifying a malicious process, the *PMon* and *FCMon* modules are anomaly based and hence probabilistically bound to miss some of the malicious activity. Also, these modules are based on the logging of IRP calls and file activity. The time lag between logging these activities and parsing them for anomalies provide a small window for the ransomware to perform its malicious activities as discussed in Section 2.5.3.

2.7 Related Work

Detection techniques. Kharraz et al. [12–14] propose systems that monitor the I/O request patterns of applications for signs of ransomware-like behaviors. Scaife et al. [16] have designed *CryptoDrop*, a system that alerts users during suspicious file activity, e.g., tampering with a large amount of the user's data. Sgandurra et al. [33] propose *EldeRan*, a machine learning approach that monitors actions performed by applications in their first phases of installation and checks for characteristics signs of ransomware. Lee et al. [46] propose a ransomware prevention mechanism based on abnormal behavior analysis in a cloud system. Cabaj et al. [47] present a software-defined networking (SDN) based detection approach that utilizes the characteristics of ransomware communication. Andronio et al. [48] propose a technique to detect Android ransomware that applies to only mobile platforms- where applications are analyzed in-depth before they are released in any app market. Huang et al. [49] propose a measurement framework for end-to-end of ransomware payments. In contrast, `RWGuard` is the fastest solution that identifies ransomware infection in real-time with decoy techniques, prevents malicious processes from making changes to the files, and also determines the original intent of file changes. Bijitha et al. [50] present a survey on ransomware detection techniques that reviews the existing solutions and validates them using specific performance metrics.

Post-encryption techniques. Kolodenker et al. [32] propose a system, called *PayBreak*, that intercepts system provided crypto functions, collects and stores the keys, and thus, can decrypt files only for the ransomware families that use system provided crypto functions.

Continella et al. [31] propose the *ShieldFS* tool that monitors low-level file system activity to model the system over time. Whenever a process violates these models, the affected files are transparently rolled back. However, it requires shadowing a file whenever it is modified and thus incurs high overhead. FlashGuard, a system developed by Huang et al. [15] leverages the fact that SSD performs out-of-place writes and thus holds the invalid pages for up to 20 days to perform data recovery after ransomware encryption. However, this type of recovery methods conflict with the idea of secure deletion and may result in privacy issues and data leakage. Given the limitations of the existing post-encryption recovery techniques, it is of uttermost importance that faster detection techniques be developed against ransomware.

Decoy techniques. Decoy techniques have been previously proposed to defend against insider threats [51]. Though some research work [11, 12] recommends using decoy files for detecting ransomware, such previous work does not include any analysis on the effectiveness of the decoy files. *Randomly generated decoy* files in commercial solutions (e.g., [52]) are susceptible of detection by sophisticated ransomware. Moreover, unlike `RWGuard`, their decoy files are deployed during the installation process which simply leaves the files unmodified for a long time and thus makes these files less interesting for the ransomware. Also, it is not clear how these solutions would handle special ransomware families, e.g., Mamba, Petya, and Matsnu, that affect only a predefined list of system files.

Cryptographic primitives identification techniques. Discovering cryptographic primitives in a given binary is another research direction where crypto-ransomware including cryptographic operations could be identified beforehand [45, 53]. Calvet et al. [53] developed such a technique and evaluated the performance of their system on a set of known malware samples. Lestringant et al. [54]'s approach to obtaining the similar goal leverages graph isomorphism techniques. Although these approaches could identify cryptographic primitives in obfuscated programs, their poor performance makes them impractical for real-time defense even with the most recent work [45] resulting in a 5-6X slowdown in average.

3. GHOSTBUSTER: A FINE-GRAINED APPROACH FOR ANOMALY DETECTION IN FILE SYSTEM ACCESSES WITH ENHANCED TEMPORAL USER PROFILES

3.1 Introduction

Insider threats represent a real danger for sensitive data owned and/or managed by organizations. Data can be compromised by malicious or compromised users within an organization. As users within organizations may have access to sensitive datasets, depending on their role or function in the organizations, data stolen or misused by either malicious or compromised insiders may result in major losses. A notable example of an insider attack is the breach [17] at Sony Pictures Entertainment, where one of the attackers was a former system administrator with a technical background and detailed knowledge of Sony's internal systems.

As discussed by Sallam et al. [55], organizations typically deploy different data protection techniques, such as authentication [56] (to verify the identity of the user trying to access the data), access control [57] (to check whether an authenticated user has permission to access specific data resources), and encryption (to protect the data at rest and while being transmitted across systems). Even though such techniques represent important building blocks for comprehensive data protection solutions, alone they are unable to offer strong protection against insider threat [4]. The reason is that insiders have permissions to access data and often have detailed knowledge about the organization's internal procedures, the location of sensitive data files and weaknesses security processes. Because of the serious threats posed by insiders to organizations, specialized protection techniques have been proposed [4, 58–61].

A key category of protection techniques is represented by anomaly detection (AD). Such techniques can identify unusual access patterns that are often typical data accesses

made by malicious or compromised insiders [18, 19]. A good protection solution is thus to deploy anomaly detection techniques alongside authentication, access control, and encryption. Several AD techniques specialized for data protection have thus been proposed, some of which are deployed at the data repository level [20, 21, 62–64] whereas others work at the network level [65–67]. However, a significant drawback of those techniques is that they either only protect data stored in relational databases, or mainly focus on network level activity patterns which may not be sufficiently fine-grained to identify malicious insider activities. However, as many applications store their data in file systems, it is critical that data access anomaly detection techniques be developed for file systems.

Some proposed AD techniques for file system accesses leverage file system features, such as file system hierarchy [68], file name, working directory, and parent directory [69]. Other AD techniques [70] rely on a file system in user-space (FUSE) to capture runtime operations. Approaches have also been proposed specifically tailored to assure file system integrity [71, 72]. The main limitation of such approaches is their inability to support fine-grained monitoring of accesses to files. For example, a mechanism only monitoring the set of files accessed by a user would not be able to detect anomalous accesses by insiders as long as the user has the read permission on the files. For example, consider the case of an employee of an organization that for his daily tasks accesses only 30% of the records in a file and therefore has read permission on this file. Now, if the employee all of a sudden accesses 90% of the file records, such access is anomalous with respect to the access pattern expected for his daily tasks. Even though there could be reasons for such anomalous access, it is critical that such access be promptly detected and flagged as anomalous. Another limitation of existing AD solutions is the high false positive rates when new files or new users added to the system [73, 74].

In this chapter, we propose an effective approach to detect anomalous file system accesses by malicious or compromised insiders. Our approach has two phases: the *Profile Creation (PC)* phase and the *Anomaly Detection (AD)* phase. The first phase is used to collect fine-grained information about the file accesses resulting from a user's regular file system activities and to create a fine-grained user profile by using a combination of dif-

ferent techniques. We also characterize the user profiles according to the time dimension by extensively analyzing the timestamp of the file accesses and thus build enhanced temporal user profiles. These user profiles are later used in the AD phase to monitor the file system usage and raise alerts upon identification of anomalous file access. Unlike previous approaches [68, 75], our AD system can classify access activities to new files dynamically added to the system. We are also able to automatically create profiles for new users dynamically added to the system- independent of whether there is a role-based access control mechanism exists in the system. We leverage unsupervised approaches to cluster the existing users and to identify the cluster for the new user.

Challenges: Unlike AD techniques for relational databases [20, 21] where SQL queries provide a structure to represent and learn a user's normal access patterns, in file systems the task of representing access patterns is challenging due to the lack of semantic information about file accesses. Moreover, the profiles need to be accurate in order to minimize false negatives or false positives. A large number of false positives, i.e., non-anomalous accesses classified as anomalous may disrupt the users' normal activities and require the intervention of security staff for further analysis of the detected anomalies. On the other hand, a large number of false negatives, i.e., undetected anomalous accesses, undermine data protection. The trade-off between the organizational impact of false positives and security is a significant challenge in creating useful profiles [22]. Also, an AD technique must not significantly impact data access times. To address the issue of false positives, our approach is based on extracting a minimal set of interesting features from the users' block level file access information in the *PC* phase. In the next step, these features are used to build the fine-grained user profiles. In the *AD* phase, our solution uses a set of distance functions to measure the difference between a user's profile (i.e., expected behavior) and his file accesses at runtime (i.e., observed behavior). Our performance evaluation results show that our approach is able to identify the following types of anomalies: anomalous access size, anomalous access frequency, and anomalous access pattern.

Contributions: Our contributions can be summarized as follows:

- A block level access anomaly detection mechanism, which to the best of our knowledge, is the first to model users' file access patterns at such a fine granularity.
- A set of efficient algorithms that: (1) extract relevant features from the users' block level access information; (2) build accurate user profiles also including temporal file access information; and (3) identify anomalous accesses at runtime. Our algorithms also handle new files and new users in the system without requiring the *PC* phase to be run again.
- A taxonomy of types of anomalous file access and a detailed accuracy comparison of our approach with other approaches based on this taxonomy.
- An extensive experimental evaluation of our fine-grained file system AD mechanism on data collected from an organizations' file repository accesses. The evaluation demonstrates that our approach achieves an accuracy of 98.7% in detecting anomalies while incurring an overhead of 2%.

3.2 Preliminaries

This section provides some background on the notions which are related to the rest of the contents of the chapter, e.g., blktrace and episodes.

3.2.1 Blktrace Utility

Blktrace is a Linux kernel utility which is used to trace block layer I/O operations. Whenever a file is accessed, blktrace fetches the access information at the block level and denotes this access as an *event*. Another utility implemented in the Linux kernel which formats these events obtained from the blktrace utility is called *blkparse*. Blktrace utility has a reasonably low overhead of only 2%. Furthermore, blktrace is highly configurable and can easily be customized to return particular events, e.g., only write operations or only read operations. The following is an example of an event returned by blktrace:

8,0	0	427	8.06743	57768	I	R	1729336	+32	[gedit]

Table 3.1.: Blktrace event components

427	represents the event's sequence number
8.06743	represents the event's timestamp
R	represents a read operation
1729336	represents the sector number from where the operation starts reading
+ 32	represents the number of sectors read, i.e., 32 sectors or 4 blocks are read including the sector number 1729336 (1 block = 4096 bytes = = 8 sectors, 1 sector = 512 bytes)

The event components that are highlighted in the example are the ones we use for our profiling. A brief explanation of these components is given in Table 3.1.

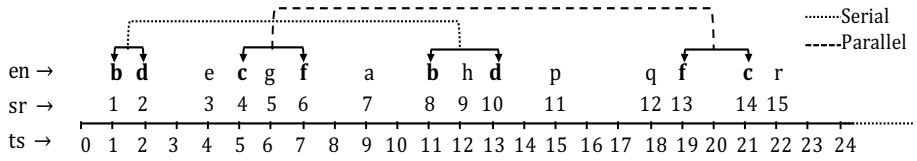


Figure 3.1.: Event sequence E

3.2.2 Event Sequences and Episodes

While the files are accessed in a system, blktrace collects the block level access information from the OS kernel and returns a sequence of events. The events are ordered with their timestamp values, and the sequence numbers are ordered as well (ascending order). Let E be a sequence of n events: $\{e_1, e_2, \ldots, e_n\}$. A blktrace event comprises of a number of attributes as discussed in Section 3.2.1. For the purpose of simplicity, we use an example here where each event consists of only three attributes: en (name of the event); sr (sequence number of the event); and ts (timestamp of the event). Therefore, we represent e_i, i.e., the ith event, as (en_i, sr_i, ts_i). Hence, the n events' names are: $en_1, en_2, \ldots, en_n$;,

sequence numbers are: sr_1, sr_2, ..., sr_n,, and the timestamps are: ts_1, ts_2, ..., ts_n, respectively. Note that, $sr_i \leq sr_{i+1}$, and $ts_i < ts_{i+1}$ for $i = 1, 2, \ldots, n-1$, since the events are ordered by sequence numbers and timestamps. An event sequence consisting of $n = 15$ events, denoted with E, is shown in Fig. 3.1.

To identify correlations among events and to discover their patterns, we use the notion of episode [76]. An episode is a set of events (not necessarily in a consecutive manner) that are likely to occur together. Hence, when a user is interacting with the file system, the episodes that frequently occur in the blktrace events represent a user's independent tasks. An episode is called a *serial* episode if the episode's events maintain a consistent order. Otherwise, we call the episode a *parallel* episode. Fig. 3.1 shows an example where event d occurs after event b twice in the sequence which means that they are correlated. Therefore, the b and d events generate a serial episode $\alpha = \{b, d\}$ with occurrences: $\{(b, 1, 1), (d, 2, 2)\}$ and $\{(b, 8, 11), (d, 10, 13)\}$. An example of parallel episode in Fig. 3.1 can be $\beta = \{c, f\}$ with occurrences: $\{(c, 4, 5), (f, 6, 7)\}$ and $\{(f, 13, 19), (c, 14, 21)\}$. Note that there is no restriction on the order of the events in this episode.

Consider two serial episodes $\alpha = \{a, c, d\}$ and $\gamma = \{a, b, c, d\}$. Here, α becomes a subepisode of γ since episode α's events are nothing but a *subsequence* of the events in episode γ. We represent this subepisode relation as $\alpha \sqsubset \gamma$ (also, γ is a superepisode of α). Again, let $\alpha = \{d, b, c\}$ and $\gamma = \{a, b, c, d\}$ be two parallel episodes. Here, episode α is a subepisode of episode γ since episode α's events are a *subset* of episode γ's events.

Table 3.2.: Anomalous file accesses' taxonomy along with a comparison among access control mechanism (AC), a file level profiling approach (FLP), e.g., [68], and our fine-grained profiling approach (FGP)

Case #	Anomaly Cases	Attack Model	AC	FLP	FGP
1	Access request to a file without user permission	Masquerade attack	✓	✓	✓
2	Anomalous clusters of file accesses	Masquerade attack	×	✓	✓
3	Anomalous frequency of access clusters	Masquerade and insider attacks	×	×	✓
4	Anomalous access size	Data harvesting attacks (insider)	×	×	✓
5	Anomalous access segment	Data harvesting attacks (insider)	×	×	✓
6	Anomalous frequency of file access	Data harvesting attacks (insider)	×	×	✓

3.3 A Taxonomy of Anomalous File Accesses

This section presents a taxonomy for different cases of anomalous file accesses. Table 3.2 summarizes six such cases. Furthermore, based on this taxonomy, Table 3.2 compares our fine-grained profiling approach with other existing approaches, i.e., we compare how these approaches would handle such anomalous cases. The approaches in comparison are: access control mechanism, a file level profiling approach (e.g., [68]) and our fine-grained profiling approach. The columns *AC*, *FLP*, and *FGP* indicate the detection capability of these approaches, respectively.

Case 1: Anomalous File Access w/o Permission. When an attacker attempts to read-/write a file that is not accessible for the original user, this it raises this anomaly case. A masquerader who steals the credentials of the legitimate user, e.g., by phishing attacks, and thus logs into the system may raise this kind of anomalies since the masquerader does not know the legitimate user's file permissions. Note that all mechanisms in this comparison can detect this anomaly.

Case 2: Anomalous Access Clusters. An advanced attacker who has knowledge about the legitimate user's file access permissions may access only those files to which the user has permissions. However, such an attacker may not access the files in the same way the legitimate user accesses. Therefore, the access clusters would be different. Hence, a masquerader (possesses the legitimate user's credential) without the knowledge of the valid user's file access behavior would raise this anomaly case. Note that the access control technique only checks for access permissions and thus cannot detect this anomaly case whereas both Gates et al. [68] and our fine-grained approach can identify this anomaly case since these approaches monitor the legitimate user's access clusters beforehand and learn the profiles.

Case 3: Anomalous Frequency of Access Clusters. In many situations, this is normal that an insider within an organization knows the access permissions of other users (especially the ones under his hierarchy) and also the files that are accessed by the legitimate user in normal situations. When such an insider intends to steal some information while re-

maining within the regular file accesses, he may access a specific set of clusters repeatedly (with the notion that the data he intends to steal can be obtained through only those set of files). As a result, a specific set of clusters are accessed with a higher frequency which is anomalous with respect to the normal behaviors. In order to identify this anomaly, we need advanced temporal user profiling (details in Section 3.4.4) which is provided by only our approach among the others in comparison.

Case 4: Anomalous Size of File Accesses. An insider within an organization who is logged into his account may perform data harvesting attacks by reading more blocks from a confidential file than usual while accessing the same clusters stored in his profile with regular frequency. It is impossible to detect such data harvesting attack without deploying a fine-grained block-level profiling approach and as shown in Table 3.2, no approach except ours can identify such anomalies (details in Section 3.4.2).

Case 5: Anomalous Segment of File Accesses. While accessing files with regular access sizes (to avoid the anomaly case 4 above), an insider may read unusual segments of a file and thus perform data harvesting attack. As shown in Table 3.2, only we are able to identify this anomaly case (details in Section 3.4.2).

Case 6: Anomalous Frequency of File Accesses. An insider who avoids detection for anomaly cases 4 and 5 may access a file with an elevated frequency. While the other approaches in comparison fail, our approach is able to identify this anomaly case (details in Section 3.4.4).

3.4 Profile Creation (PC) Phase

The Profile Creation *(PC)* phase comprises of four different software modules as highlighted in Fig.3.2. Apart from these four modules, the *PC* phase uses the blktrace tool to fetch the I/O requests' queue from the kernel space. This phase runs when the user is accessing the file system during regular activities. The rest of the section describes the threat model and the details of the four modules.

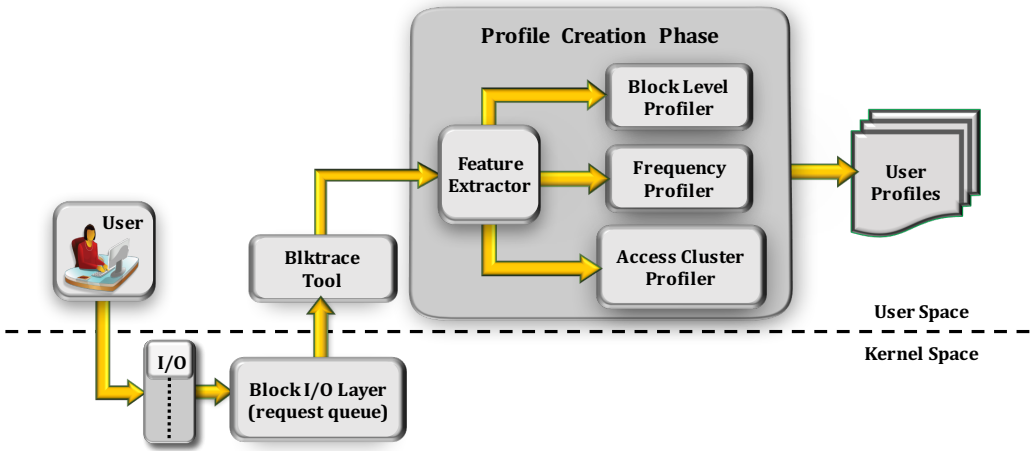


Figure 3.2.: Profile Creation *(PC)* phase architecture

Threat Model: In our threat model, we consider an adversary that intends to perform masquerade (cases `1, 2,` and `3` in Table 3.2) and data harvesting (cases `3, 4, 5,` and `6` in Table 3.2) insider attacks against a file system. We design three categories of adversary: (1) a non-expert attacker gaining access to the file system but having no prior knowledge of the normal file accesses by the users, (2) a medium attacker scenario where the attacker has already made accesses and 25% of these accesses are anomalous, (3) a knowledgeable attacker scenario where only 3% of the accesses by the attacker are anomalous. However, the performance of our *AD* phase does not significantly vary with the varying adversary scenarios (further details are given in Section 3.6). This is due to the fact that we monitor each blktrace event individually and thus do not depend on the number of anomalous blktrace events. Moreover, we compute an anomaly score for each user and compare the score with a threshold (further details are given in Section 3.5). A user is prevented from accessing the files only after her anomaly score exceeds the set threshold. The adversary may plan to evade the detection by remaining within the set threshold while stealing an insignificant amount of data, e.g., by exfiltrating only one file that the insider usually has access to. However, our solution would still be able to detect such an exfiltration if-

- the file is generally accessed within a context (e.g., along with other related files) but at the time of exfiltration it is accessed without any context (case `2` in Table 3.2), or
- the access size is anomalous (case `4` in Table 3.2), or

- the access segment is anomalous (case 5 in Table 3.2), or
- the file is accessed with anomalous frequency (case 6 in Table 3.2).

We would like however to point out that our solution, as many other security solutions, should be used in combination with other techniques for in-depth data protection. More specifically to have a stronger security level, in addition to our technique, one should also deploy techniques such the ones proposed in [64, 77]. Those techniques create profiles concerning the use of data once the user has gained access to these data. For a malicious insider to succeed in exfiltrating the data, the insider would have to copy/move/e-mail the data. By using techniques in [64,77], we can detect anomalies with respect to these actions. Therefore, by combining our solution with the ones in [64, 77] one can not only detect anomalies in the data accesses from files but also anomalies in the use of the data once returned to the user.

We consider the operating system to be trusted. Moreover, since our *AD* solution runs on the user space, we leverage the Software Guard Extensions (SGX, available with Intel Processors) to ensure the security of the user profiles.

3.4.1 Feature Extraction *(FE)*

In the following, we first discuss the blktrace event features that we use for user profiling.

- **User ID**: This is the ID (e.g., username) of the file system user which identifies him uniquely. For each I/O request that results in an event from blktrace, the id of the user account responsible for the I/O request is represented by this feature.
- **File ID/name**: Blktrace returns a sector number which is later translated to block number and then to inode to identify the file uniquely. This feature thus represents the id of the file. Note that if the user performs a write operation on a file, it may change the sector numbers. Therefore, if the translation is performed at the end of the tracing by blktrace module, it is not possible to identify the correct file names.

In order to address this issue, we translate sectors to the file names online, i.e., while the blktrace module is running.

- **Access type**: If the I/O request resulting in the blktrace event is for a read operation, the type feature is presented by 'R'. For a write operation, the type feature is 'W'.

- **Access size**: This feature for a blktrace event represents the number of blocks accessed as the result of the I/O request.

- **Segment of file**: With this feature, we aim to identify the segment of the file that is usually accessed by the user. In other words, we identify the relative position of the accessed blocks in the file memory. For instance, assume that a file f has 20 blocks in the memory and the first 5 blocks are accessed by an I/O operation. This feature represents the segment as `[0,0.25]`. In contrast to our fine-grained approach, the existing approaches that profile at file level can only identify if a file, which is not accessed in a normal situation, has been accessed maliciously. Capturing the accessed segment of a file has some advantages over capturing the block numbers since the block numbers may change later, for example, after a write operation, or after disk defragmentation.

- **Timestamp**: This feature represents the time when the I/O request has been placed. This feature is useful to build fine-grained temporal profiles of the users.

- **Sequence number**: This feature is useful to identify the spatial properties of a user's file activities, e.g., how frequently a file is accessed among a number of I/O requests, whether there is any other file which is accessed frequently along with a particular file.

As mentioned earlier, the blktrace tool returns one event for each I/O request. The *FE* module converts these blktrace events to E, a sequence of events. An event $e \in E$ is represented as *(uid, fname, atype, sz, sg, ts, sr)*, where *uid* denotes the user id, *fname* denotes the file name, *atype* denotes the type of the access, *sz* denotes the access size, *sg* denotes the accessed segment of a file, *ts* denotes the timestamp of the event, and *sr* denotes

the event's sequence number. Given that E consists of n events, it can be represented as follows:

$$\begin{aligned} E &= \{e_1, e_2, \ldots, e_n\} \\ &= \{(uid_1, fname_1, atype_1, sz_1, sg_1, ts_1, sr_1), \\ &\quad \ldots, (uid_n, fname_n, atype_n, sz_n, sg_n, ts_n, sr_n)\} \end{aligned}$$

1: **Input**: E, L_{ts}, L_{sr}, $minSp$
2: **Output**: $\mathcal{C}$
3: $m \leftarrow 0$
4: $Cand_1 \leftarrow$ set of distinct events
5: while $Cand_{m+1} \neq \emptyset$
6: $\quad Freq_{m+1} \leftarrow findFreq(E, Cand_{m+1}, L_{ts}, L_{sr}, minSp)$
7: $\quad m \leftarrow m + 1$
8: $\quad Cand_{m+1} \leftarrow genCand(Freq_m)$
9: $\mathcal{C} \leftarrow Freq_1 \cup Freq_2 \cup \ldots \cup Freq_m$
10: return $\mathcal{C}$

Algorithm 1: Discovering access clusters $\mathcal{C}$

3.4.2 Block Level Profiling *(BLP)*

In order to understand the user accesses at block level, this module stores a map from the users to the files with the following features: ***uid, fname, sz,*** and sg. For each $(uid, fname)$ pair, this module then computes the following: the average access size sz_{avg}, maximum access size sz_{max}, and the standard deviation of the access sizes sz_{sd}. Moreover, this module leverages the segment feature sg to understand whether the user accesses random segments of the file. For instance, if e_i and e_j are two blktrace events that represent accesses to the same file $\mathbf{f}$, the *BLP* module checks if sg_i and sg_j segments overlap in the file memory or if the segments come from random locations of the file memory. Also, it helps to understand whether the user is always accessing the new data appended to the file. We use a parameter named $rand$ bit and set it to 1 if the segments accessed from a file are random, otherwise, this bit is set to 0. Given the set $\mathbf{F}$ which represents all the files in the system, and a file $\mathbf{f} \in \mathbf{F}$, this component profiles a user $\mathbf{u}$'s block access as

$B_{\mathbf{f}} = \{sz_{avg}, sz_{max}, sz_{sd}, rand = 0/1\}$. For all such files, the block level profiling is represented as $\mathcal{B} = \langle B_1, B_2, \ldots, B_{|\mathbf{F}|}\rangle$ where $|\mathbf{F}|$ is the number of files in the system.

3.4.3 Access Cluster Profiling *(ACP)*

When a single task performed by some file system user involves accessing a number of files, whenever the user performs that task, a specific set of files are accessed together. This set of files form a cluster which we represent with an episode. As a result, the frequency of such episodes essentially reflects the frequency of the corresponding tasks. We represent such an episode ε as $C_\varepsilon = \{\varepsilon, cf, S/P\}$ where cf denotes the frequency of the cluster and S/P denotes the type of the episode (i.e., serial/parallel). The episodes identified for a user are finally stored as $\mathcal{C} = \langle C_{\varepsilon 1}, C_{\varepsilon 2}, \ldots, C_{\varepsilon m}\rangle$ where m represents the number of episodes.

While identifying the access clusters, we enforce that the events within an episode are not too scattered- neither in terms of the time difference nor in terms of the sequence number difference. This ensures that we do not consider those clusters of accesses as episodes where the events in the clusters are not related enough. To denote the maximum time difference between the events in an episode, we use a threshold L_{ts}. Similarly, we use a threshold L_{sr} that denotes the maximum sequence number differences between two events. For instance, two events a and b can belong to the same episode only if $|ts_b - ts_a| < L_{ts}$ and $|sr_b - sr_a| < L_{sr}$. We use another threshold $minSp$ (i.e., minimum support) to discard the episodes that are not frequent. When an episode's frequency surpasses this minimum frequency requirement, that episode is classified as frequent.

Algorithm 1 presents the steps to compute the second component $\mathcal{C}$ of a user profile. In this algorithm, the first iteration considers every unique event to be a candidate ($Cand_1$) itself. In the next iterations, larger candidates are computed to be considered as frequent episodes. For instance, if the size of the frequent episodes obtained in some iteration is m (which means the set $Freq_m$ computed from $Cand_m$ using the `findFreq` algorithm), in the next iteration we compute $Cand_{m+1}$, i.e., the set of candidates for $m+1$ size episodes from $Freq_m$ (using the `genCand` algorithm).

Note that we have used the following Apriori [78] property to perform these computations:

Lemma 1 *Let β and α be episodes s.t. $\alpha \sqsubset \beta$. If β is frequent, α is also frequent.*

Hence, we use the idea that an $m+1$ size episode can be a candidate only if all of this episode's m size subepisodes are frequent. As a result, we avoid a number of $m+1$ size episodes (for which all of its m size subepisodes are not frequent) and reduce the overall computation overhead by not computing frequency for those.

We describe the steps of `findFreq` and `genCand` algorithms in the following.

Frequency Computation (`findFreq`): Depending on the context, researchers have proposed different frequency definitions, e.g., [76, 79]. However, since we want to profile the file access frequencies by users, we use a frequency definition based on non-overlapping. In order to understand the definition, consider an episode α with two occurrences: α_1 and α_2. Now these occurrences are considered as non-overlapping if no event that belongs to α_1 appear among the events that belong to α_2 and vice versa. Let α = $\{a, b, c\}$ be a serial episode, and let $\alpha_1 = \{(a, sr_{a1}, ts_{a1}), (b, sr_{b1}, ts_{b1}), (c, sr_{c1}, ts_{c1})\}$ and $\alpha_2 = \{(a, sr_{a2}, ts_{a2}), (b, sr_{b2}, ts_{b2}), (c, sr_{c2}, ts_{c2})\}$ be two of its occurrences. The episodes α_1 and α_2 do not overlap if for all events $x \in \alpha$, $sr_{x1} < sr_{a2}$ and $ts_{x1} < ts_{a2}$, or, for all events $x \in \alpha$, $sr_{x2} < sr_{a1}$ and $ts_{x2} < ts_{a1}$. If E is a sequence of event such that-

$$E = \{(b,1,2),(d,2,5),(a,3,6),(c,4,7),(f,5,10),$$
$$(c,6,12),(d,7,13),(c,8,14),(a,9,16),(f,10,20)\}$$

for episode $\alpha = \{b, d, c\}$, the number of overlapping occurrences are two, i.e., $\{(b,1,2), (d,2,5),(c,4,7)\}$, and $\{(b,1,2),(d,7,13),(c,8,15)\}$. However, if we consider non-overlapping occurrences only, only one occurrence, e.g., $\{(b,1,2),(d,2,5),(c,4,7)\}$ could contribute towards its frequency.

The `findFreq` function in Algorithm 2 is used to compute the frequency of serial episodes. This algorithm takes E, the event sequence as input along with $Cand_i$ (i.e., the set of candidate episodes of length i), L_{ts}, L_{sr}, and $minSp$. At the end, this algorithm returns frequent episodes of size i, i.e., $Freq_i$, which is a subset of $Cand_i$.

1: **Input**: $E, Cand_i, L_{ts}, L_{sr}, minSp$
2: **Output**: $Freq_i$
3: $Freq_i \leftarrow \emptyset$
4: for each candidate episode $\varepsilon \in Cand_i$
5: $waits(\varepsilon[1]) \leftarrow waits(\varepsilon[1]) \cup (\varepsilon, 1, 0, 0)$ /*initialize $waits$ list*/
6: $\varepsilon.freq \leftarrow 0$ /* set frequency to 0 for all candidates */
7: for each event $e \in E$
8: for each cluster automaton $A_\varepsilon = (\varepsilon, j, ts_f, sr_f) \in waits(e)$
9: if $ts_f = 0$ & $sr_f = 0$ /* automaton at the first state */
10: $ts_f \leftarrow e.ts$ /* update first event timestamp */
11: $sr_f \leftarrow e.sr$ /* update first event sequence number */
12: $waits(e) \leftarrow waits(e) - A_\varepsilon$ /* remove the automaton from $waits(e)$ list */
13: if $j = L$ & $e.ts - ts_f \leq L_{ts}$ & $e.sr - sr_f \leq L_{sr}$
14: $j \leftarrow 1$ /* reached final state of automaton, reset to first state */
15: $\varepsilon.freq \leftarrow \varepsilon.freq + 1$ /* increment frequency */
16: else if $e.ts - ts_f \leq L_{ts}$ & $e.sr - sr_f \leq L_{sr}$
17: $j \leftarrow j + 1$ /* move to the next state */
18: else
19: $j \leftarrow 1$ /* e is outside of L_{ts} or L_{sr} range, reset to first state */
20: $waits(\varepsilon[j]) \leftarrow waits(\varepsilon[j]) \cup A_\varepsilon$ /* add A_ε to a new $waits$ list */
21: for each candidate episode $\varepsilon \in Cand_i$
22: if $\varepsilon.freq \geq minSp$
23: $Freq_i \leftarrow Freq_i \cup \varepsilon$
24: return $Freq_i$

Algorithm 2: Finding frequent episodes (`findFreq`)

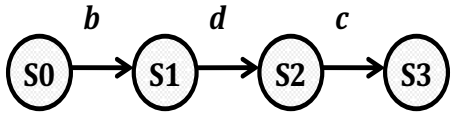


Figure 3.3.: Finite state automata A_ε for $\varepsilon = \{b, d, c\}$

We leverage the idea of finite state automata while identifying the frequent episodes. For instance, let ε represents a serial episode $\{b, d, c\}$. We define an automaton A_ε corresponding to this episode as shown in Fig. 3.3. The automaton transits from state $S0$ to $S1$ upon encountering the event b, then from state $S1$ to $S2$ with event d, and so on.

Episode ε's automaton A_ε is represented as $(\varepsilon, j, ts_f, sr_f)$ when it is ready to move to the jth state. Here, ts_f represents the timestamp of the first event in the automaton whereas sr_f is the sequence number of that first event. The $waits()$ data structure keeps track of every candidate episode's automaton's next expected event. For instance, $waits(e)$ lists all the automata that would move to their next states with event e. Hence, upon the occurrence of event e, all automata in $waits(e)$ proceed to their next states. If automaton $A_\varepsilon = (\varepsilon,$ $j, ts_f, sr_f) \in waits(e)$, ε episode will move to its jth state if it encounters an occurrence of event $e = (fname_e, atype_e, sg_e, sz_e, ts_e, sr_e)$ and if the conditions $ts_e - ts_f \leq L_{ts}$ and $sr_e - sr_f \leq L_{sr}$ are satisfied. If ε is a parallel episode, we use a similar algorithm but it works a little differently from the one used for serial episodes. Each entry in the list $waits(e)$, is an ordered pair like, (α,j), which now indicates that there is a partial occurrence of parallel episode α which still needs j events of type e before it can become a complete occurrence. For example, parallel episode α = $\{a, a, b, c, c\}$ will initially have three lists, namely, $waits(a)$, $waits(b)$, and $waits(c)$, and they will have entries (α,2), (α,1), and (α,3), respectively.

1: **Input**: $Freq_m$
2: **Output**: $Cand_{m+1}$
3: $Cand_{m+1} \leftarrow \emptyset$
4: $S_m \leftarrow sort(Freq_m)$
5: for each episode $s_{mi} \in S_m$
6: for each episode $s_{mj} \in S_m$ where $j \geq i$
7: if $S_m.msg[i] = S_m.msg[j]$
8: for $k = 1$ to m
9: $\zeta[k] \leftarrow s_{mi}[k]$ /* generate new candidate */
10: $\zeta[m+1] \leftarrow s_{mj}[k]$
11: for each $\alpha \sqsubset \zeta$ where $|\alpha| = m$
12: if $\alpha \notin Freq_m$ /* if any α is not frequent */
13: goto 6 /* ζ is also not frequent, start over */
14: $Cand_{m+1} \leftarrow Cand_{m+1} \cup \zeta$
15: return $Cand_{m+1}$

Algorithm 3: Candidate generation (`genCand`)

Candidate Generation (`genCand`): We use the `genCand` function in Algorithm 3 to generate the parallel episode candidates. Since there is no particular order of the events in parallel episodes, we sort the events according to their $fname$ feature. For instance, we store the episode $\varepsilon = \{d, a, f\}$ as $\{a, d, f\}$. Each episode $\varepsilon \in Freq_m$ is sorted in this way and is then stored in S_m. We perform further sorting in S_m for efficiency. For example, episode $\{a,\ d,\ f\}$ is stored after episode $\{a,\ d,\ e\}$. We denote an episode with size m residing in the ith index of S_m as $S_m[i]$. Note that, if episodes $S_m[i]$ and $S_m[j]$ have first $m-1$ events in common, i.e., they differ only in their mth event, they belong to a maximal similarity group which we denote as *msg*. We store the *msg* for each episode in S_m (e.g., $S_m.msg[i]$ stores the index of $S_m[i]$'s *msg*'s first episode). In other words, the msg array points to the first episode in the maximal similarity group which improves the efficiency of the algorithm.

Computational Complexity of Automata

In the *PC* phase, we reduce the overhead of computing the candidate episodes' frequencies by using finite state automata. Instead of scanning all the events in E for computing the frequency of each candidate episode, our algorithm needs to scan the events in E only once for all the candidate episodes. Upon scanning an event e in E, all the candidate episode automata that can accept e advance to their next states. Moreover, one candidate needs exactly one automaton which also reduces the overhead of the *AD* phase.

Addition of New Files

Whenever a new file is added to the system (at runtime) that was not present during profile creation, we need to classify access to this file accurately to avoid false positives/negatives. In our solution, we leverage the location and type information of the new file for this classification. The location often indicates the purpose of the file and its similarities with the neighboring files, e.g., files under a single source code repository.

In the case of access to a new file, we consider the current finite state automata and compute the distance between the new file and the file expected to be accessed. In Fig. 3.3, consider the case that a new file *g* is accessed after file *b*, i.e, when the automata is at state $S1$. Since *g* is a new file, we compute the distance between *g* and the expected file *d* in terms of their distance in the file hierarchy. We identify the least common ancestor (LCA) of these two files in the hierarchy and compute its distances from the two files in comparison. We normalize this total distance by the worst case scenario distance where the LCA is itself the root directory of the file system. The formula to compute the anomaly value is the following:

$$anom\_val(g,d) = \frac{dist(g, LCA(g,d)) + dist(d, LCA(g,d))}{dist(g, ROOT) + dist(d, ROOT)} \tag{3.1}$$

If the LCA for *g* and *d* is the ROOT, the $anom\_val$ becomes 1 which represents that these two files have nothing in common other than being in the same file system. On the other hand, an insignificant $anom\_val$ denotes that the files' distance is smaller and their probability of being similar is high. Moreover, if also the types of these files are same, we conclude that the *g* and *d* files are similar and profile the new file at runtime which accelerates classifying the future accesses to the new file.

Addition of New Users

Along with handling new files dynamically, it is also important that an AD system is able to profile new users at runtime. Depending on whether the organization has a role-based system, there are two possibilities:

1. If there exists no role-based access control in the system, we can use an unsupervised learning approach [80–84] to mine the user roles. A role is primarily defined by the permissions that are granted to the user. Therefore, when there is no semantic information available other than the user permissions, we can use unsupervised machine learning algorithms to cluster the users that have a similar set of permissions. Although traditional data mining algorithms aim to identify non-overlapping clus-

ters, in our case, we need to allow different roles (i.e., identified different clusters) to have overlapping permissions. This distinction from traditional clustering makes the problem of role mining more challenging. However, we leverage existing technique, i.e., [80] to perform such role mining for a new user when there is no role-based access control mechanism deployed in the system.

2. If a role-based access control mechanism already exists in the system, we leverage techniques, e.g., [85], that assign a new user to an existing role or create a new role if required.

Profiling Benign Activity Changes by the Users

Note that, the statistical fingerprint of the users' benign activities can change and shift (i.e., concept drift) over a period of time due to changes in user activity at the organization or even due to some system updates. In the cases in which a role-based access control system is deployed, we address these concept drifts of benign activities by observing if the benign activities are changing for other users within the same role. If this is the case, we assign a lower anomaly score for such unknown file activities. If a single user's activities change significantly whereas no other user under that role requests similar accesses, our solution rapidly increases the anomaly score of that user (detailed discussion on anomaly scores are given in Section 3.5) and the user is eventually flagged.

If there is no role-based access control system deployed, techniques that integrate supervised machine learning and control theoretic model for detecting concept drift, such as, [86], can be leveraged. We leave this as a future work. Moreover, adversarial concept drifts that intend to avoid detection by traditional concept drift detection techniques can be handled using solutions, e.g., [87], that leverage adversarial forethought and incorporate the context into the drift detection task.

3.4.4 Frequency Profiling *(FP)*

For each file, this component computes the temporal and spatial frequencies of the user's accesses. This module uses the following features from blktrace events: *uid, fname, ts,* and *sr*, and then computes for file $\mathbf{f}$ its spatial frequency, $fr^s_{\mathbf{f}}$, and temporal frequency, $fr^t_{\mathbf{f}}$. The frequency profiling consisting of both these frequencies is accumulated as $fr_{\mathbf{f}}$. Thus the profile component is $\mathcal{F} = \langle fr_1, fr_2, \ldots, fr_{|\mathbf{F}|} \rangle$.

Finally, a user profile is formally represented as $\mathcal{P} = \langle \mathcal{B}, \mathcal{C}, \mathcal{F} \rangle$ (see Sections 3.4.2 and3.4.3 for components $\mathcal{B}$ and $\mathcal{C}$, respectively).

Spatial frequency: To compute the spatial frequency, we first need to set an interval (e.g., an interval of 100 file accesses, independent of time) and then count the number of accesses to different files in that interval. We consider a sliding interval of size SI for this purpose. While we use a fixed interval for spatial frequency, we compute enhanced temporal profiles by leveraging a multi-level profiling approach as described in the following.

Temporal frequency: Although many existing profiling approaches, e.g., [75], use a fixed time interval approach for computing temporal frequency, we employ a multi-level temporal profiling technique for obtaining our fine-grained user profiles. The resulting enhanced temporal profiles not only improve the detection accuracy but also removes ambiguity from both of the profiling and anomaly detection phases. To understand how this multi-level time granularity approach works, let A and B be two tasks performed by the user (the tasks represent episodes α and β, respectively).

1. A: This task is performed only on one day in a week (on any arbitrary weekday), but is performed for 5 times on that day (i.e., with a frequency of 5).

2. B: This task is performed on each of the weekdays but the frequency for each day is only 1.

In the following, we compare two approaches for temporal profiling: (1) fixed time interval approach and (2) our multi-level time granularity approach.

Fixed Time Interval Approach

Similar to the spatial frequency computation, a simple idea for temporal frequency could be setting a fixed interval of time, TI, and compute the frequencies in this interval. Due to the various frequencies of different file accesses, there are also many options for setting the time interval TI, e.g., one hour, one day, one week, or even one month. Hence, the time granularity for the profiles can vary from very fine-grained (e.g., one minute) to very coarse-grained (e.g., one year). For files that are accessed very frequently, one may choose to compute the files' access frequencies for all these granularity levels. Again, for a file which is accessed very rarely, computing its frequency at a coarse-grained level (e.g., with TI = one month) could be sufficient.

Now, with this fixed time interval approach, the user tasks presented above, i.e., A and B, would have identical frequencies when the value of TI is set to one week. In contrast, if the value of TI is set to one day, the tasks would have different frequencies. The frequency of task A would be 5 for an arbitrary weekday and 0 for the rest of the weekdays whereas the frequency of task B would be 1 for all the weekdays. Now if TI is set to be one week (because if TI is set to be one day, it would not be possible to profile task A since this task is performed 5 times on only one weekday), there is no way of differentiating the frequencies for the tasks A and B, and thus it creates ambiguity in the profile. This also impacts the accuracy of the profile, e.g., this fixed time interval approach would result in a false negative when an insider performs task A only once on a weekday.

Multi-level Time Granularity Approach

In this approach, we use a multiple granularity temporal structure, TS, instead of using a plain time interval TI. An example of TS is the following: if a user performs a task on all the days of only an arbitrary month of every year, the corresponding TS would store the temporal profile as ($\forall_{Year}$, $\exists_{Month}$, $\forall_{Day}$). Fig. 3.4(a) and 3.4(b) represent the TSs of tasks A and B, respectively. Let the length of time for the user logs considered in these structures be one month. We show only three levels, i.e., months, weeks and days,

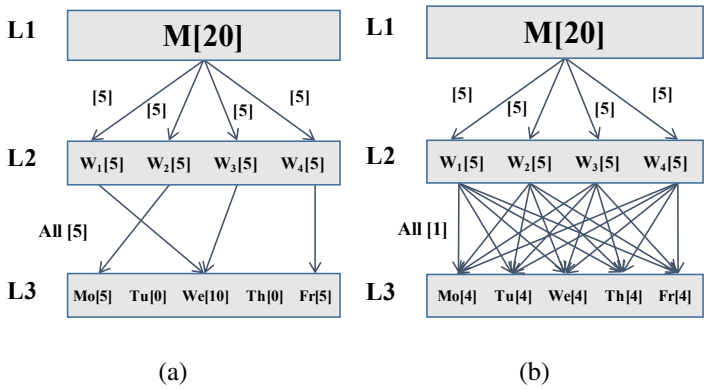


Figure 3.4.: Profiling (a) task A, and (b) task B

with the level identifiers $L1$, $L2$, and $L3$, respectively. According to the descriptions of the tasks, they have identical frequencies for the levels $L1$ (frequency is 20 for both) and $L2$ (frequency is 5 for both). The only difference in frequency they have is at level $L3$. According to Fig. 3.4(a), task A is performed on different days in different weeks since this is performed on arbitrary weekdays. To profile the arbitrary frequency of A on different weekdays, our multi-level time granularity approach creates the following TS: ($\forall_{Month}[20]$, $\forall_{Week}[5]$, $\exists_{Day}[5]$). The $\forall$ symbol denotes *for all*, the $\exists$ symbol represents the fact that the event exists only once, and the '[]' symbol denotes the exact frequency at different levels. Again, according to Fig. 3.4(b), task B is performed on every weekday for all the weeks considered in the experiments. Hence, the TS for this according to our multi-level time granularity approach would be ($\forall_{Month}[20]$, $\forall_{Week}[5]$, $\forall_{Day}[1]$). From the above discussion, it is evident that this approach can distinguish the different temporal frequencies of different tasks without introducing any ambiguity unlike a naive fixed time interval approach.

We can profile even more complicated file access temporal behaviors with such multi-level approach. Consider a scenario where a specific task is performed on Tuesday and on Thursday with a frequency of 1 for each. Our temporal structure represents this task as ($\forall_{Month}[8]$, $\forall_{Week}[2]$, $\forall_{Day}\ \langle 1, 5\rangle\ [1, 1]$). Note that, the numbers in $\langle\rangle$, i.e., 2 and 4, represent Tuesday and Thursday, respectively. With $\forall_{Day}$ symbol before $\langle 2, 4\rangle$ the temporal structure means that for all days in $\langle\rangle$, this rule applies. Moreover, with $[1, 1]$, the structure shows the frequencies for the days.

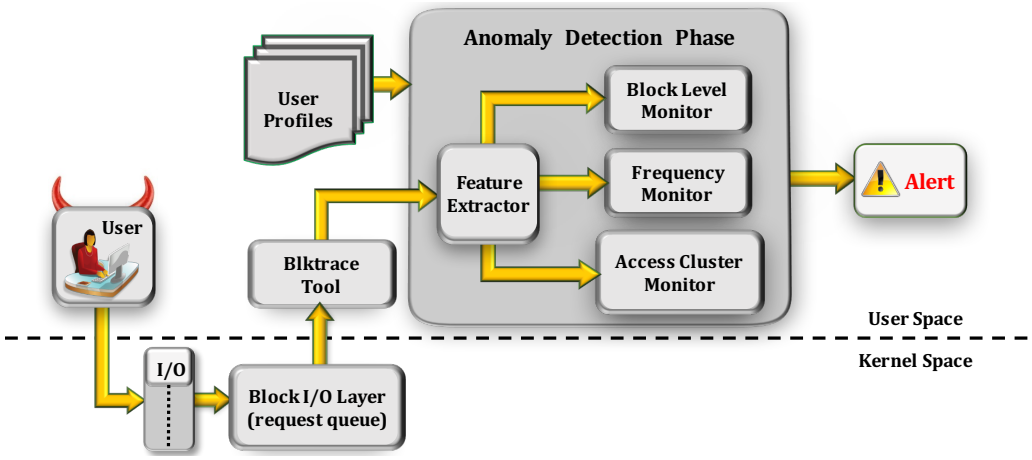


Figure 3.5.: Anomaly Detection *(AD)* phase architecture

3.4.5 User Profiles' Storage

A critical requirement for our solution is that the user profiles need to be securely stored to prevent their tampering as tampered profiles may undermine our AD solution.

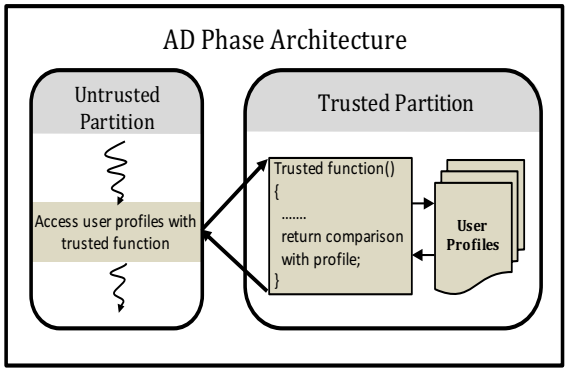


Figure 3.6.: AD phase partition into trusted and untrusted parts

As our AD solution runs on the user space, in order to protect the user profiles, we leverage the Software Guard Extensions (SGX) available with Intel Processors [88]. SGX provides enclaves, i.e., isolated memory regions, which enable us to protect certain sensitive data (user profiles) and the code that operate on this data. Security and privacy of such enclaves are achieved by hardware support [88].

As shown in Fig. 3.6, we partition the code of our AD solution into two parts: a trusted part, and an untrusted part. The trusted part contains the user profiles and the code portion

that needs to work with the user profiles. The rest of the code resides in the untrusted part. Whenever there is a need to compare the user's behavior with the profile, a trusted function is called to enter the enclave, where it can access the user's profile in plain text. Note that the enclaves reside in an encrypted memory and they are considered trusted because they cannot be modified after they have been built. In our setting, only the AD process can access the enclave memory whereas all other attempts to access enclave memory from outside the enclave are denied by the processor, even if these attempts are by some privileged users. As soon as an enclave is maliciously modified (either by a user or a malicious software), the CPU is able to detect it. This protects the user profiles in the enclave from being exposed even if the application, Operating System, or BIOS are compromised.

Since enclaves are stateless, all their contents (user profiles) are lost when the enclaves are destroyed (enclaves can be destroyed when, e.g., the system goes to sleep or if the AD process exits). Therefore, to preserve the user profiles, we make a copy of these profiles, encrypt them, and store them outside the enclave in the untrusted memory. Though this copy (or, multiple copies) resides in the untrusted memory, the encryption itself provides assurances of confidentiality, integrity, and authenticity on these user profiles.

3.5 Anomaly Detection (AD) Phase

The tasks performed by this phase's *Feature Extractor (FE)* module is same as that of the *PC* phase's *FE* module. Note that, the *User Profiles* that are generated in the previous phase are used in the *AD* phase as an input. The rest of the modules in the *AD* phase (Fig. 3.5) monitor the user's runtime file accesses and for each user compute a shared parameter W_{AF} to weigh his anomalous behaviors. If the value of W_{AF} exceeds a set anomaly threshold, $minAnom$, the user's accesses are classified as anomalous, and his account is flagged. The modules raise different anomaly flags for different anomalous file access activities and update the W_{AF} value accordingly. **Note that, the *AD* phase does not interrupt the normal I/O operations of the user until the W_{AF} exceeds the set**

threshold $minAnom$. A user is prevented from accessing the resources only after W_{AF} value becomes greater than $minAnom$.

3.5.1 Block Level Monitoring *(BLM)*

The *BLM* module of the *AD* phase monitors the sizes of accesses to the files. For a file, if the access size is larger than the file's corresponding sz_{max} or if the access size is not within the following range: $[sz_{avg} + \delta_1 * sz_{sd}, sz_{avg} - \delta_1 * sz_{sd}]$ (where δ_1 is an adjustment parameter and can be any real number), the *BLM* module raises an anomaly flag AF_1. To determine how much a particular anomaly flag increases the value of W_{AF}, we use different distance functions, for example, if the size of an access (sz) extracted from the corresponding blktrace event is greater than the sz_{max} value stored in the profile, we measure the distance function as $dist(AF_1) = \frac{(sz - sz_{max}}{sz_{max}})$. Moreover, if the user accesses random segments of the file while the value of $rand$ is 0 in the user profile, anomaly flag AF_2 is used to represent this event.

3.5.2 Access Cluster Monitoring *(ACM)*

In this module, we use finite state automata to keep track of the access clusters (that are discovered in the *PC* phase as described in Section 3.4.3). Assume that a user requests an I/O operation for a file **f** to which he does not have access permission. An anomaly flag AF_3 is raised in such scenarios. However, if the user has permission, the *ACM* module exploits the fact that at least one automaton (access cluster) would accept the resulting blktrace event and would move to the next state. If no automaton accepts the event, this module raises an anomaly flag AF_4. It is possible that a single event is accepted by multiple automata and all these automata move to their next states. If the access clusters have anomalous frequencies, the *ACM* module raises an anomaly flag AF_5. Note that we consider only the superepisodes for efficiency purpose. For instance, if episode $\{a, b, c, d\}$ is identified as frequent, we do not create automata for its subepisodes, e.g., $\{a, b\}$ or $\{a, b, c\}$ and a single automata is created to consider the superepisode itself and also its subepisodes. In the case

Table 3.3.: Mapping between the anomaly cases and flags

Anomaly Case #s	Anomaly Flags	Module (AD phase)
1	AF_3	*ACM*
2	AF_4	*ACM*
3	AF_5	*ACM*
4	AF_1	*BLM*
5	AF_2	*BLM*
6	AF_7	*FM*

of a new file, we compute its distance with the existing files as described in Section 3.4.3 and use the computed $anom\_val$ as an anomaly flag AF_6.

3.5.3 Frequency Monitoring *(FM)*

If the user accesses a file f with a frequency that exceeds δ_2 * $fr_{\mathbf{f}}$ (where δ_2 is another adjustment parameter and can be any positive real number), the *FM* module AF_7 anomaly flag. Note that the spatial frequency is computed over a number of accesses and this is independent of time interval.

With the enhanced temporal profiles $\{\mathcal{T}, \mathcal{T}_{\mathcal{DS}}\}$, where the first part stores the discovered tasks and the second part stores the tasks' T_{DS}s, the *FM* module identifies any anomaly that do not conform with the profile. For instance, from Section 3.4.4, the T_{DS} of task B is ($\forall_{Month}[20]$, $\forall_{Week}[5]$, $\forall_{Day}[1]$). Hence, the *FM* module would raise an alarm if this task is not executed on a working day or it is executed more than once on a working day. Note that unlike other fixed time interval approaches that introduce ambiguity, our *FM* module is able to monitor the A task independently. A mapping between different anomaly cases (as presented in Section 3.3) and the anomaly flags is given in Table 3.3.

3.6 Performance Evaluation

In this section, we explain our experiment setup, present the evaluation metrics and the performance of our AD mechanism for different anomaly cases along with a comparison with the other existing approaches.

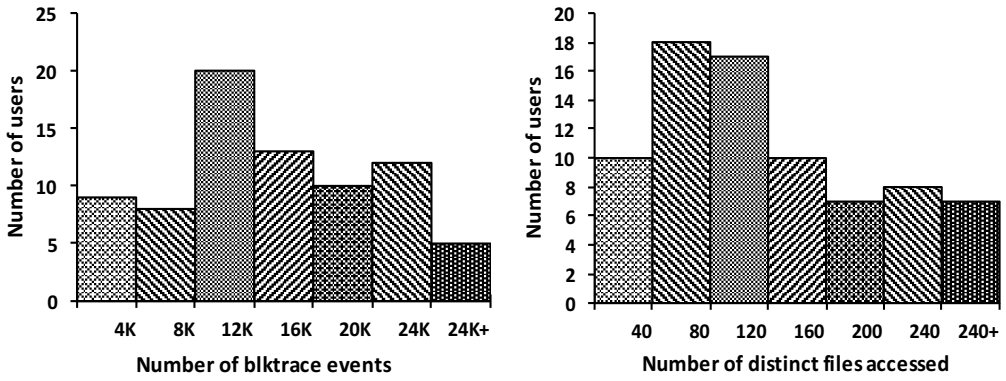
3.6.1 Experiment Setup and Evaluation Metrics

For the experiment setup, we use a file repository from Wikipedia which consists of 560 files. We run blktrace for a time length of 2 months to capture the block level accesses by 77 users. In order to evaluate the case of new users, we use the access logs of 70 users in the *PC* phase for profiling and add the remaining 7 users' access logs at runtime. Moreover, to measure the accuracy of the $anom\_val$ computation for new files, we create 20 new files in the system at runtime and add one access to each file (where half of the accesses are benign and the remaining half of the accesses are malicious). The blktrace events from the first month are used in profiling (i.e., *PC* phase) and blktrace events from the second month are used in the *AD* phase to evaluate the performance of our anomaly detection system.

Fig. 3.7(a) shows a histogram that presents the number of blktrace events in ranges or different number of users. For instance, 20 users among the 77 result in 8000 to 12000 blktrace events in the duration of two months. Fig. 3.7(b) shows a similar histogram with respect to the number of users but this time presents the number f distinct files accessed by the users instead of the number of blktrace events. For instance, 10 users among the 77 access 0-40 distinct files in the duration of two months. Note that, according to the logs collected in our experiment, we have observed that a single file access, on average, can result in six blktrace events.

According to our observation, the *PC* phase parameters L_{ts}, L_{sr}, and $minSp$ play an important role in improving the accuracy of the *AD* phase. On one side, setting too small values to these parameters results in failure to identify correlations of different accesses whereas on the other side, setting too large values result in identification of too many correlations that are not relevant. In order to make a balance, we choose $L_{ts}tobe$20 minutes and L_{sr} to be 10 which result in an optimal accuracy. Moreover, we set different $minSp$ values for different iterations of the profiling phase to accommodate different candidate sizes.

In order to evaluate the performance of the *AD* phase, we synthetically generate anomalous access requests (like previous work that synthetically generated datasets, e.g., [89]) for



(a) (b)

Figure 3.7.: Different ranges of (a) blktrace events, and (b) distinct file accesses by the users

each anomaly case (anomaly cases 1-6 in Table 2) and inject those in the users' file access logs. We create four test datasets:

1. Test dataset I (`TS-I`): We use the access logs of the second month directly for this test dataset to represent a benign scenario.

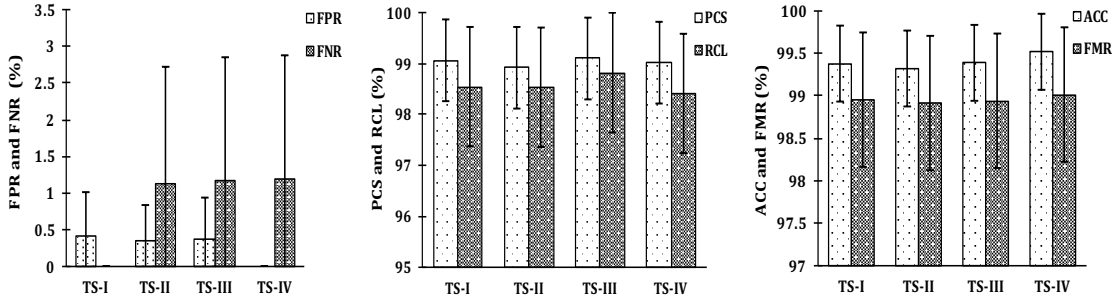
2. Test dataset II (`TS-II`): We remove only 3% of the logs in `TS-I` and insert blktrace events that are anomalous with respect to the profile to represent a knowledgeable attacker scenario.

3. Test dataset III (`TS-III`): We remove 25% of the logs in `TS-I` and insert blktrace events that are anomalous with respect to the profile to represent a medium attacker scenario.

4. Test dataset IV (`TS-IV`): In `TS-IV`, the access logs are generated by accessing the files completely randomly to represent a non-expert attacker.

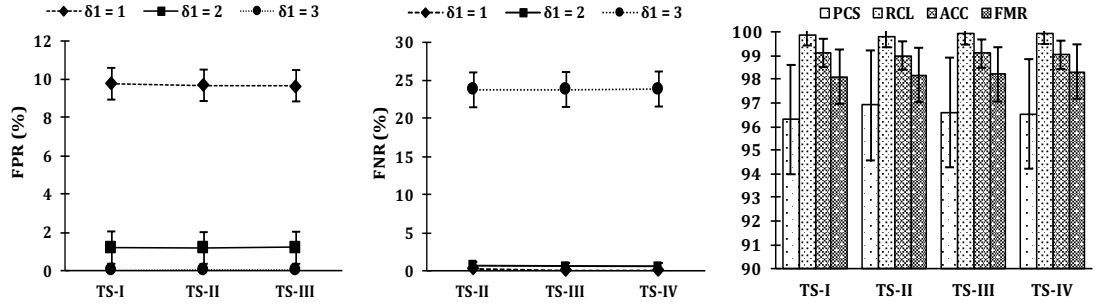
Note that we use our multi-level data structure approach for generating temporal user profiles (instead of the fixed time interval approach) in all the experiment results shown in this section. A performance comparison between these two approaches is given in Section 3.6.3.

We perform our experiments with an Intel(R) Core (TM) i7-6700 CPU machine consisting of two 3.40 GHz cores and 16GB memory. We use Ubuntu-14.04 as the operating system.



(a) (b) (c)

Figure 3.8.: ACM module: mean (a) FPR and FNR, (b) PCS and RCL, (c) ACC and FMR values with confidence interval of standard deviation



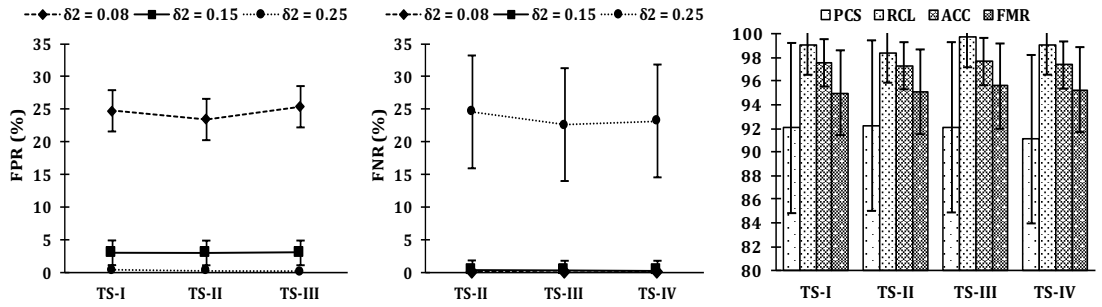
(a) (b) (c)

Figure 3.9.: BLM module: mean (a) FPR, and (b) FNR for different δ_1, (c) PCS, RCL, ACC, and FMR for δ_1=2 with confidence interval of standard deviation

Metrics: A false positive (FP) arises when a benign file access is evaluated as anomalous, whereas a false negative (FN) means that an anomalous file access is evaluated as a benign access. The true positives (*TP*) and true negatives (*TN*) represent correct evaluations of anomalous and normal accesses, respectively. The metrics we use for our performance evaluation are given below:

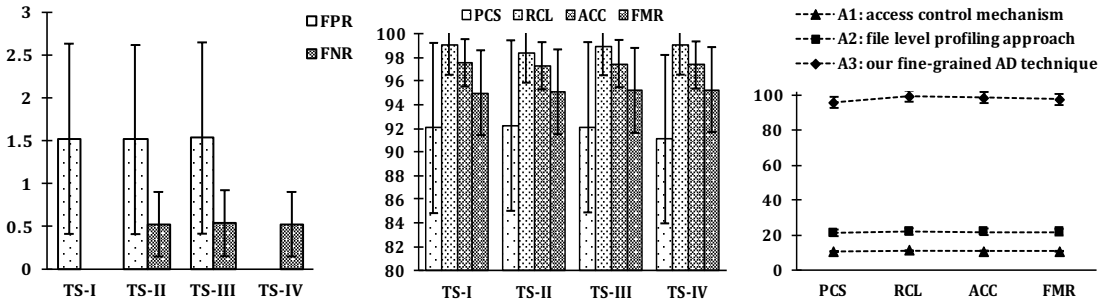
- False positive rate (FPR) = $\frac{FP}{(FP+TN)}$

- False negative rate (FNR) = $\frac{FN}{(FN+TP)}$
- Precision (PCS) = $\frac{TP}{(TP+FP)}$
- Recall (RCL) = $\frac{TP}{(TP+FN)}$
- Accuracy (ACC) = $\frac{(TP+TN)}{(TP+TN+FP+FN)}$
- F-measure (FMR) = $\frac{2TP}{(2TP+FP+FN)}$



(a) (b) (c)

Figure 3.10.: FM module: mean (a) FPR, and (b) FNR for different δ_2, (c) PCS, RCL, ACC, and FMR for $\delta_2 = 0.15$ with confidence interval of standard deviation



(a) (b) (c)

Figure 3.11.: Combined (a) FPR, FNR, (b) PCS, RCL, ACC, FMR of our approach, (c) Comparison with existing techniques

3.6.2 Experiment Results

In this section, we classify the anomaly cases described in Section 3.3 into three separate attack scenarios. These three scenarios present anomaly cases [1,2,3], [4,5], and [6], respectively. For each attack scenario, we generate different versions of `TS-II, TS-III,` and `TS-IV` test datasets.

ACM Module For Anomaly Cases 1-3

For this attack scenario, we generate the `TS-II, TS-III,` and `TS-IV` test datasets so that the modified blktrace events include the following cases: I/O requests to files that the user does not have access permissions, I/O requests to files that have no correlation among them within a short sequence/temporal interval, and I/O requests that represent accessing some clusters of files with elevated frequency.

As given in Table 3.3, the anomaly cases 1-3 are detected by the *ACM* module. The *FPR* and *FNR* of this module are presented in Fig. 3.8(a). Note that we show the *FPR* for only the test datasets `TS-I, TS-II, TS-III` whereas we show the *FNR* for only the test datasets `TS-II, TS-III, TS-IV`. This is because all the events in `TS-IV` are anomalous (since `TS-IV` is generated randomly) and thus cannot have *false* positives. On the other hand, all the events in `TS-I` are benign (since these are taken from the original logs) and these cannot have *false negatives*.

The results show that the *FPR* is lower than *FNR* for all the test datasets. This is because the probability of no automata accepting a file access in the *ACM* module is negligible since the resulting event needs only one automaton to consider the access as benign. In contrast, if the file access in consideration is actually malicious, this may still be accepted by an automaton of the *ACM* module and might not be flagged as malicious resulting in a false negative. However, in most of the cases, the automata that accept this malicious access would eventually fail to transit to their next states and thus identify the file access as malicious.

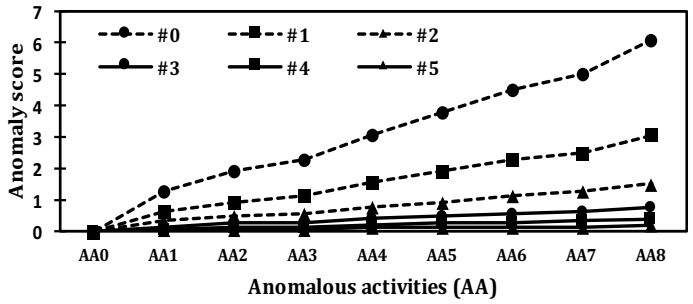


Figure 3.12.: Concept drift experiments with multiple anomalous activities (previously unknown) when between 0 and 5 other users perform similar file accesses

Finally, the *FPR* and *FNR* of the *ACM* module (in average) are 0.38% and 1.17%, respectively. According to our observation, the performance of the *ACM* module does not vary significantly with the varying test datasets. This is due to the fact that this module monitors each blktrace event individually and thus does not depend on the percentage of the anomalous blktrace events in the test datasets.

The *PCS* and *RCL* values for this module are shown in Fig 3.8(b) which have an average of 99.1% and 98.82%, respectively. The *ACC* and *FMR* values shown in Fig. 3.8(c) have averages of 99.39% and 98.94%, respectively.

Experiment on concept drifts of benign activities: We perform this experiment for the case when there exists a role-based access control system. Changes in activities by a single user (within a role) raises anomaly flags as mentioned in Section 3.5. Within a role consisting of 10 users, we show in Fig. 3.12 how the concept drifts of eight anomalous activities ($AA0 - AA8$) affect the anomaly scores of the users. We show the changes in anomaly score for a user when between 0 and $50 - 5$ other users within the same role perform similar new (i.e., previously unknown) activities. If no other user (i.e., $\#0$ in Fig. 3.12) performs similar new activities, the anomaly score of that user increases significantly after eight anomalous tasks. If one other user (i.e., $\#1$ in Fig. 3.12) performs similar activities, the anomaly scores for both users increase at a lower rate. Finally, if five other users (i.e., $\#5$ in Fig. 3.12) perform similar tasks, the change in the anomaly score for all six users become almost negligible.

BLM Module For Anomaly Cases 4-5

For this attack scenario, we generate the `TS-II, TS-III,` and `TS-IV` test datasets so that the modified blktrace events include the following cases: I/O requests with anomalous access sizes and I/O requests with anomalous access segments.

As given in Table 3.3, the anomaly cases 4 and 5 are detected by the *BLM* module. The *FPR* and *FNR* of this module are presented in Fig. 3.9(a) and Fig. 3.9(b), respectively. While evaluating the *BLM* module, we set different values to δ_1: 1, 2, and 3, to observe how this parameter impacts the results. Setting a smaller value to δ_1, e.g., 1, means that the acceptable range for the access size is reduced to [$sz_{avg} + sz_{sd}$, $sz_{avg} - sz_{sd}$] which in turn flags a number of benign accesses as malicious resulting in a 9.66% *FPR* in average. In contrast, setting a larger value to δ_1, e.g., 3, means that the acceptable range for the access size is increased to [$sz_{avg} + 3 * sz_{sd}$, $sz_{avg} - 3 * sz_{sd}$] which in turn fails to flag a number of malicious accesses resulting in a 23.81% *FNR* in average. Hence, we set the value of $\delta_1 = 2$ which balances the two extreme results with high *FPR* and *FNR*, and results in 1.19% *FPR* and 0.11% *FNR*. In contrast to the *ACM* module, the *FPR* of this module is higher than the *FNR*. This is due to the fact that this module analyzes the blktrace events in a more fine-grained level and thus in few cases flags the benign accesses as malicious. On the other hand, this module can identify even knowledgeable attackers with fine-grained attack models resulting in a negligible *FNR*. Note that some accesses in the `TS-IV` test dataset conform to the access sizes stored in the profiles. Hence, while we expect all accesses in `TS-IV` to be malicious (since these are generated completely randomly), we observe some false negatives. The *PCS*, *RCL*, *ACC*, and *FMR* values for this module are shown in Fig. 3.9(c) which have an average of 96.5%, 99.89%, 99.08%, and 98.2%, respectively.

FM Module For Anomaly Case 6

For this attack scenario, we generate the `TS-II, TS-III,` and `TS-IV` test datasets so that the modified blktrace events include the case of I/O requests with anomalous spatial and temporal frequencies.

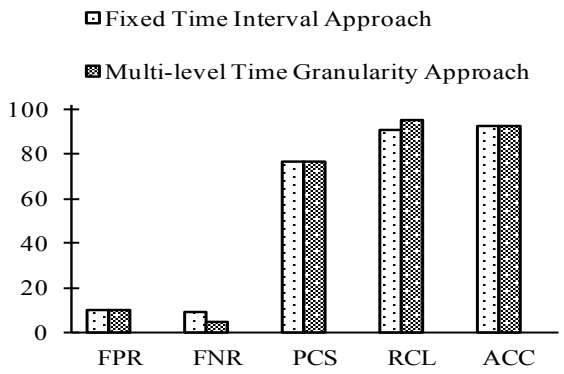


Figure 3.13.: Results for Multi-level Temporal Profiles

As given in Table 3.3, the anomaly case 6 is detected by the *FM* module. The *FPR* and *FNR* of this module are presented in Fig. 3.10(a) and Fig. 3.10(b), respectively. While evaluating the *FM* module, we set different values to δ_2: 0.08, 0.15, and 0.25, to observe how this parameter impacts the results. Setting a smaller value to δ_2, e.g., 0.08, means that the acceptable range for the frequency is reduced which in turn flags a number of benign accesses as malicious resulting in a high *FPR*. In contrast, setting a larger value to δ_2, e.g., 0.25, means that the acceptable range for the access size is increased which in turn fails to flag a number of malicious accesses resulting in a high *FNR*. Hence, we set the value of $\delta_2 = 0.15$ which balances the two extreme results with high *FPR* and *FNR*, and results in 3.02% *FPR* and 0.33% *FNR*. The *FNR* for the test dataset `TS-IV` is due to the same reason as described in the previous section. The *PCS*, *RCL*, *ACC*, and *FMR* values for this module are shown in Fig. 3.10(c) which have an average of 92.09%, 99.67%, 97.64%, and 95.57%, respectively.

3.6.3 Multi-level Temporal Profiles

In this section, we compare the performances of the fixed time interval approach and our multi-level time granularity approach as described in Section 3.4.4. To generate the malicious dataset, we modify the access logs in `TS-I` so that it has anomalous frequency for the tasks discovered by the *ACP* module. Fig. 3.13 shows the comparison between these two temporal profiling approaches in terms of their *worst case performances*. It compares

the two approaches' *FPR*, *FNR*, *PCS*, *RCL*, and *ACC* values. The results show that while the maximum *FNR* for the fixed time approach is $\sim 9.3\%$, using the multi-level temporal profiles decreases the worst-case performance to $\sim 4.65\%$. This results in an increased *RCL* from $\sim 90.69\%$ to $\sim 95.34\%$. While the multi-level temporal profiles are able to identify anomalous accesses that the fixed time approaches cannot (i.e., decreases *FNR*), the worst-case *FPR*, *PCS*, and *ACC* values remain unchanged due to a case where the number of false positives is maximum.

3.6.4 Comparison with Existing Approaches

In order to compare our solution with the other existing approaches, we build a comprehensive test dataset where we gather the anomalous datasets from Sections 3.6.2, 3.6.2, and 3.6.2 each having identical numbers of anomalous logs.

Fig. 3.11(a) shows the final *FPR* and *FNR* values of our *AD* approach on this combined anomalous dataset (1.53% and 0.53% in average, respectively). The *PCS*, *RCL*, *ACC*, and *FMR* values are presented in Fig. 3.11(b) which are 95.92%, 99.46%, 98.7%, and 97.57%, in average, respectively.

In Fig. 3.11(c), we compare among the following three mechanisms: access control ($A1$), an approach that profiles at the file level ($A2$ [68]) and our AD solution ($A3$). We compare the performance of these approaches in terms of their *PCS*, *RCL*, *ACC*, and *FMR*. For instance, the accuracies for these approaches are: $A1 : 10.82\%$, $A2 : 21.92\%$, and $A3 : 98.7\%$. The explanation behind such huge difference is that the $A1$ solution can only identify the first anomaly case, $A2$ can only identify the cases 1 and 2 whereas our solution $A3$ is able to identify all the anomaly cases in consideration.

According to this comparison, we can conclude that even having the solutions $A1$ and $A2$ in a combined fashion is not sufficient to identify a large variety of malicious file system activities. Hence, more fine-grained monitoring, for instance, $A3$ is required to protect the file systems from both malicious insiders and outsiders.

3.6.5 Overhead Analysis

The *PC* phase of our AD solution which builds the user profiles executes off-line and thus does not have any impact when the users are interacting with the file system. However, since we collect the blktrace events during the users' regular file system activities, the only overhead it incurs is the overhead by the blktrace tool itself which is negligible (2% [90]). The space overhead of the *PC* phase necessarily depends on the time length of the data collection and also on the volume of the file system activities by the users (if the data is collected for a longer time, the volume of the training data naturally increases). Moreover, as per our experiments, the space overhead for the user profiles is reasonable (< 1 MB on average). The *AD* phase is executed by an independent host with an Intel(R) Core (TM) i7-6700 CPU machine which monitors the blktrace events fetched by the blktrace tool and checks them against the stored user profiles with the added overhead of SGX. Again, it incurs a CPU overhead of 2% at runtime to the users.

3.7 Related Work

Most of the existing anomaly detection mechanisms have been proposed either for networked systems [65–67] or for relational databases [20, 21, 62, 63]. Debar et al. [91] introduce a taxonomy that defines families of intrusion-detection systems according to their properties whereas Glass-Vanderlan et al. [92] present a survey of intrusion detection systems leveraging host data. An anomaly-based IDS for modern mobile devices is presented by Damopoulos et al. [93]. In [94], Vrakas et al. propose a technique to protect against spoofing attacks, e.g., masquerading and identity theft at real-time. Nguyen et al. [95] present a technique for generic feature selection in order to perform intrusion detection. Baracaldo et al. [96] extend the RBAC (role-based access control) model in terms of the trust the system has on its users and also with a risk assessment process. In [97], Görnitz et al. develop an anomaly detection methodology that achieves higher accuracy while requiring less labeled data. In order to enhance the performance of anomaly detection, some research work [98, 99] suggest building robust temporal user profiles with different time

granularities. Srivastava et al. [100] propose Verity where they use a block-chain network to detect insider attacks.

In [101], Bowen et al. deploy decoys to identify insiders within an organization and also develop a method for automatic decoy injection [102]. Gates et al. [68] propose a method to detect information theft by insiders where they compute the correlation between a resource/file and the users accessing that file. They leverage the file system hierarchy for this purpose. As a result, this approach cannot handle file systems that change dynamically (where files can be added, deleted, or moved) due to such dependency on the file system hierarchy. Stolfo et al. [69], having similar limitations as that of Gates et al. [68] leverage different features of the file system, e.g., file location, parent directory, current directory, etc., to identify anomalous file system accesses. In contrast, our solution is not dependent on the hierarchy of the file system and therefore, applies to file systems that change dynamically. While the anomaly detection system by Mehnaz et al. [75] cannot profile the new files or new users dynamically, our solution profiles the new files by computing its similarity with existing neighboring files and also profiles the new users either based on their roles or by monitoring their initial file access activities. Moreover, our approach computes multi-level temporal profiles to decrease the false negatives whereas the approaches [68, 75] use a fixed time interval approach to build temporal profiles.

In order to detect malicious activities from authorized insiders, Ray et al. [103] propose a framework that utilizes an attack tree. Senator et al. [104] combine different structural and semantic information from the file system and monitor user activity to identify anomalous accesses learned from suspected scenarios of a malicious insider. In [105], Claycomb et al. present a technique using directory virtualization that monitors a number of systems deployed in an enterprise simultaneously and identifies any malicious insider activity. In contrast, different application's anomalous run-time operations can be detected by Huang et al. [70]'s proposed unsupervised learning approach. For each running application, they monitor its file access activities and create a baseline profile. These application profiles are then utilized at runtime to measure the probability of the file access requests to be benign. Camiña et al. [106] avoid monitoring every single file system object and thus propose a

task-based masquerader detector. Compared to these approaches that work at a higher level of abstraction, our approach detects a broader set of anomalies by leveraging low-level access information at the block level.

While our focus is on file system confidentiality, there is also some previous research work that provides solutions for file system integrity [71,72]. I^3FS [72] is such a tool that computes the cryptographic checksums of the files, compares with the expected checksums to validate the files' integrity, and thus detects any malicious modifications at real-time.

4. PRIVACY-PRESERVING REAL-TIME ANOMALY DETECTION USING EDGE COMPUTING

4.1 Introduction

Advances in sensors (embedded systems, IoT, etc.) and wireless technologies have enabled us to collect huge amounts of data which are then analyzed in real-time to support efficient and effective decision making in many safety critical application domains, such as home security, patient monitoring, detecting cyber attacks in nuclear power plants, etc. The underlying technique used for such real-time decision making is *anomaly detection* which allows one to identify anomalous patterns that do not conform to the *expected behavior of a system* or to the *expected data a system collects/generates*. For instance, identifying an emergency situation of a patient when signals from her health monitoring devices seem anomalous or detecting anomalous trends in industrial data that are indicative of a pending system failure require effective real-time anomaly detection. Internet giants such as Google [5], Microsoft [6] and specialized companies (e.g., Anomaly [7]) are already offering such anomaly detection as a service for real-time data or for predictive maintenance.

Motivation. Even though sensor-equipped devices are able to collect large amounts of data, they are often unable to perform computationally intensive analytics, *especially in real-time*, due to their resource-constrained nature, e.g., limited computation and storage capabilities. Also, because sensor devices may not be equipped with comprehensive tools for protection against failures, in many solutions [107], storing data at a cloud/edge server is critical for minimizing data losses. In order to understand the pros and cons of such different scenarios, we summarize the results of a case study in Table 4.1. We have used a Raspberry Pi 3 to represent a sensor device, an Intel Core i7 machine as the edge server, and an Amazon AWS EC2 (t2.xlarge) instance as the cloud server (the cloud server is located in Oregon whereas the Raspberry Pi 3 and the edge server are both located in Indiana). We

execute a windowed Gaussian anomaly detection algorithm on a UCI Machine learning repository [108] dataset[1] consisting of 100000 data points, and record the latency and communication cost for different scenarios as given in Table 4.1. In scenario 1, the Raspberry Pi 3 is responsible for both anomaly detection computation and storage, i.e., the data does not leave the source and thus there is no need for data encryption to preserve the privacy. However, as mentioned above, such devices have limited computation and storage capabilities and thus cannot cope with the ***velocity*** and ***volume*** of the data, respectively, while still providing real-time support for anomaly detection. Moreover, there are many security vulnerabilities [109], e.g., weak credential/session management, impersonation, poor physical security, etc., in the deployment of such devices [110] that may compromise/wipe off the locally stored data. Though this storage limitation is solved in scenario 2 where the device encrypts the data and sends it to the cloud, the communication latency is impractical. This communication latency can be significantly reduced by sending the encrypted data to an edge server as represented by scenario 3. Note that, edge servers are as untrusted as the cloud [111, 112] and thus it is critical that the data still be encrypted if edge servers are used instead of cloud servers for data storage. However, unfortunately, even in scenario 3, the computation latency *at the device* is still impractical to support real-time processing of data acquired with high velocity. For example, if the incoming data rate (collected/generated by sensors) is 500 data points per second, the sensor can no longer catch up with the data since in that case the maximum latency that can be practical for real-time analysis of 100000 data points is 200 seconds. Scenario 4 represents the case where the device transfers the data in plaintext to an edge service provider and outsources both anomaly detection computation and storage tasks to the edge. Though scenario 4, opted by Google [5], Microsoft [6], Anomaly [7], could result in low latency, it does not take privacy into account [113]. While the data in transmission could be encrypted by the underlying transport layer protocol (e.g., SSL/TLS) to protect against man-in-the-middle attacks, the data offloaded to the edge remains in plaintext to support analytics. Such a solution is not adequate for privacy-sensitive applications, such as personal health-care systems and critical indus-

[1] http://archive.ics.uci.edu/ml/datasets/Pseudo+Periodic+Synthetic+Time+Series

Table 4.1.: Case study for different scenarios.

#	Scenario	Latency (in seconds)	Communication cost (in MB)	Privacy	Storage
1	All computation on device (no transfer of data)	716.56	0	✓	×
2	All computation on device, cloud only stores encrypted data	6590.53	25.3	✓	✓
3	All computation on device, edge only stores encrypted data	853.57	24.9	✓	✓
4	Both computation and storage at edge (on plaintext data)	125.48	24.9	×	✓

trial monitoring systems, whose data cannot be shared with an untrusted third party (i.e., edge/cloud). This work thus addresses the following research question: *is it possible to develop a privacy-preserving framework to enable efficient real-time anomaly detection on sensitive, time series, streaming data by sharing the anomaly detection computation task between the device and the edge (i.e., utilizing the computation power of the edge server without compromising the data privacy)?*

Challenges. In general, when the data is encrypted before off-loading, the edge is unable to process the encrypted data efficiently. Most approaches based on Secure Multi-party Computation (SMC) include complicated techniques, such as Yao's garbled circuits [23] and oblivious transfer, that are impractical for real-time analytics. Therefore, many existing solutions for privacy-preserving anomaly detection rely on trusted third parties or assume the presence of non-colluding third parties, thus introducing weak links in the security chain [24], whereas solutions that perturb data for privacy [25] are susceptible to data reconstruction [26]. Solutions assuming co-operative anomaly detection [27] or crowd-sourcing [28] using differential privacy do not directly apply to our scenario where there is a single data owner willing to outsource the anomaly detection task while simultaneously preserving the privacy of the data. Moreover, solutions leveraging differential privacy would require injecting a significant amount of fake data which in turn would reduce the utility significantly.

Proposed solution. We consider an *honest but curious* model for the edge, i.e., the edge service provider follows the protocol but would like to gain knowledge about the data. Since the edge is untrusted, we encrypt the data rather than using any other susceptible techniques, e.g., perturbation. In order to ensure efficiency, we utilize a lightweight encryption scheme, `Trident`, which is also semantically secure, to encrypt the data before

off-loading the data to the edge. Moreover, this scheme is optimized for aggregation operation in the encrypted domain by the usage of a telescoping series (where partial sums eventually have a fixed number of terms after cancellation). As a result, the scheme accelerates the processing of streaming encrypted data at the edge.

We leverage the *windowed Gaussian anomaly detector*— the most widely used anomaly detection technique [29, 114] that computes the probability of a data being anomalous by observing the distribution of a window of previous data points. Since this original anomaly detection algorithm operates only on plaintext data, we design a privacy-preserving version of this algorithm where the anomaly detection task is shared between the device and the edge while only the insignificant portion of the computation that the edge cannot perform (due to not having the decryption key) is carried out at the resource-constrained device. Our framework enables privacy-preserving detection of anomalies of all three categories: *point, contextual, and collective anomalies*. The proposed framework does not compromise any utility, i.e., the accuracy of our framework is same as that of the analytics on plaintext data. **Our solution results in a latency of `145.52` seconds and communication cost of `30.9` `MB` for the experiment described in the case study of Table 4.1 while preserving the privacy, and satisfying both storage and accuracy requirements simultaneously.**

The contributions of this work are the following:

1. We use a lightweight, semantically secure, and aggregation optimized (optimization is due to a telescoping series) encryption scheme dubbed as `Trident`.

2. We design a privacy-preserving framework that detects point, contextual, and collective anomalies in streaming large-scale data. We consider the windowed Gaussian anomaly detector algorithm for detecting anomalies and propose a privacy-preserving version of this algorithm. In our solution, the sensitive data is never stored in plaintext. The usage of our lightweight `Trident` scheme results in practical latency and communication overhead without compromising the accuracy of the anomaly detection process.

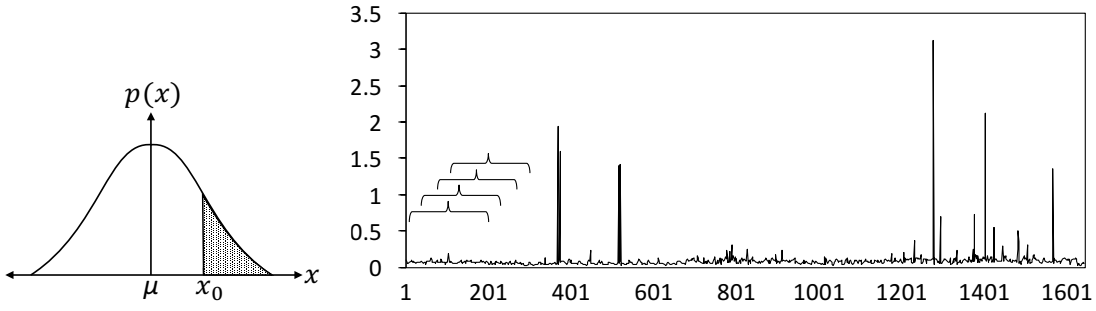
3. We evaluate the performance of the proposed framework in terms of privacy, accuracy of the resulting model, communication cost, and latency, and compare it with a

baseline scenario where the data is off-loaded to the edge in plaintext and the edge performs anomaly detection on plaintext data (i.e., scenario 4 in Table 4.1). We also perform a scalability analysis to determine the performance of our approach as the size of the data increases.

4.2 Preliminaries

4.2.1 Q Function

The Gaussian distribution has two parameters: a mean denoted by μ and a standard deviation denoted by σ. The bell shaped curve in Figure 4.1(a) represents a Gaussian distribution. The probability density function of a Gaussian distribution is given by: $p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.



(a) (b)

Figure 4.1.: (a) Gaussian distribution, (b) RealAdExchange data.

The probability that a Gaussian random variable $X \sim N(\sigma, \mu^2)$ exceeds x_0 is evaluated as the area of the shaded region shown in Figure 4.1(a) which is computed as:

$$Pr(X \geq x_0) = \int_{x_0}^{\infty} p(x)dx = \int_{x_0}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \tag{4.1}$$

By change of variables method, we substitute $y = \frac{x-\mu}{\sigma}$ and then equation (4.1) can be re-written as:

$$Pr\left(y \geq \frac{x_0 - \mu}{\sigma}\right) = \int_{\left(\frac{x_0-\mu}{\sigma}\right)}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \tag{4.2}$$

Here the function inside the integral is a normalized Gaussian probability density function $Y \sim N(0,1)$ with $\mu = 0$ and $\sigma = 1$. The integral on the right side of equation (4.2) can be termed as Q-function, which is given by:

$$Q(z) = \int_z^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \tag{4.3}$$

Hence, the relation with the Q function is:

$$Pr\left(y \geq \frac{x_0 - \mu}{\sigma}\right) = Q\left(\frac{x_0 - \mu}{\sigma}\right) = Q(z)$$

Thus the Q function gives the area of the shaded curve in Figure 4.1(a) with the transformation $y = (x - \mu)/\sigma$ applied to the Gaussian probability density function.

An error function represents the probability that the parameter of interest is within a range between $\sigma\sqrt{2}$ and $x/\sigma\sqrt{2}$ and the complementary error function represents the probability that the parameter lies outside this range. The error and the complementary error functions are given by:

$$erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-x^2} dx$$
$$erfc(z) = 1 - erf(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-x^2} dx \tag{4.4}$$

From the limits of the integrals in equation (4.3) and (4.4) it can be concluded that the Q function is directly related to complementary error function ($erfc$) by the following relation:

$$Q(z) = \frac{1}{2} erfc\left(\frac{z}{\sqrt{2}}\right) \tag{4.5}$$

4.2.2 Windowed Gaussian Anomaly Detector

The windowed Gaussian anomaly detector computes the anomaly score of a given data point by computing its probability of being normal (using the Q-function as described in Section 4.2.1) while observing the Gaussian distribution over a window of previous data points. Since the Gaussian distribution is the assumed underlying probability model in many applications that involve statistical analysis of data, the windowed Gaussian anomaly detector is the most commonly used technique for anomaly detection in such data [29,

1: $\mu = 0$ (mean), $\sigma = 1$ (standard deviation)
2: $stepSize = s$, $windowSize = 6 * s$
3: **for** $i = 0,\ldots, m-1$ (considering m data points) **do**
4: DO : collects d_i
5: DO : transfers d_i to the ESP
6: $z = \frac{(d_i - \mu)}{\sigma}$
7: $anom[i] = 1 - \frac{1}{2}\ erfc(\frac{z}{\sqrt{2}})$
8: **if** $i > 0$ **then**
9: $sum[i] = sum[i-1] + d_i$
10: **else**
11: $sum[i] = d_i$
12: **end if**
13: **if** $i > 0$ & $i\%stepSize = 0$ & $i \leq windowSize$ **then**
14: $\mu = \frac{sum[i]}{(i+1)}$
15: $\sigma = \sqrt{\frac{\sum_{k=0}^{i}(d_k - \mu)^2}{(i+1)}}$
16: **end if**
17: **if** $i > 0$ & $i\%stepSize = 0$ & $i > windowSize$ **then**
18: $\mu = \frac{sum[i] - sum[i - windowSize]}{windowSize}$
19: $\sigma = \sqrt{\frac{\sum_{k=(i-windowSize)}^{i}(d_k - \mu)^2}{windowSize}}$
20: **end if**
21: Append d_i to dataset $D = \{d_1, d_2, \ldots, d_{i-1}\}$ (for storage)
22: **end for**

Algorithm 4: Algorithm for windowed Gaussian anomaly detector on plaintext data.

114]. The window slides forward with a defined step size. The window and step sizes can be tuned to achieve better accuracy for different applications. Figure 4.1(b) plots the *realAdExchange* data [115] (consisting of 1643 data points including the anomalous ones, x axis represents data indices and y axis represents data values) which represents online advertisement clicking rates, where the metric is cost-per-click (CPC). It also shows four windows where the $windowSize$ is set to 200 and the $stepSize$ is set to 33. The μ and σ values are computed for each window sliding, i.e., each $stepSize$. **While the sensors may be able to store all the data points in a single window for analysis, note from Table 4.1 that the computation latency at the sensors is still impractical for real-time analysis.**

Algorithm 4 presents the steps of windowed Gaussian anomaly detector for plaintext data, i.e., scenario 4 in Table 4.1. The data owner DO's sensor device (in the rest of the document, the terms DO and *device* are used interchangeably) collects and sends the data points instantly to the edge in plaintext, or, in other words, without any encryption (the steps for the DO are shown in blue text). The edge service provider ESP computes the anomaly score for each data point using the current μ and σ values, and recomputes these parameters after each step size as shown in the algorithm.

4.3 Lightweight and Aggregation Optimized Encryption (TRIDENT) Scheme

In this section, we describe a lightweight and aggregation optimized encryption scheme, `Trident`, designed to optimize encryption and decryption operations, and also the aggregation operation in the encrypted domain (i.e., addition of a large amount of encrypted data). We also present a security analysis of the `Trident` scheme in Section 4.3.4 which assumes a probabilistic polynomial time-bounded adversary. Note that, there exists public key additively homomorphic encryption schemes which require expensive exponentiation operations. Since we do not need a public key, we can achieve significant performance improvements as demonstrated in Section 4.5.2.

4.3.1 Encryption Scheme

This encryption scheme can be used over any additive group, e.g., Z_ρ, where ρ is a positive integer. The scheme has three algorithms- KeyGen, EncryptData, and DecryptData as described in the following:

KeyGen: The key generation algorithm takes as input a security parameter λ and outputs a symmetric key $k \in \{0,1\}^\kappa$ for use in the encryption and decryption steps.

EncryptData: This algorithm takes as inputs a message m, the symmetric key k, and a nonce $\eta \in \{0,1\}^n$. Note that, a nonce is an arbitrary number that can be used only once in a cryptographic communication. Let $H : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \mathrm{Z}_\rho$ be a cryptographic hash

function keyed with k which outputs $H_k^\eta = H(k||\eta)$ for a given nonce η (here, $||$ denotes concatenation). The encryption of m is computed as following:

$$Enc(m, k, \eta) = m - (H_k^\eta - H_k^{(\eta-1)})$$

For simplicity, let $F(\eta, k) = F_k^\eta = H_k^\eta - H_k^{(\eta-1)}$. Therefore,

$$\begin{aligned} Enc(m, k, \eta) &= m - F_k^\eta \\ &= c \end{aligned}$$

DecryptData: This algorithm takes as input a ciphertext c and key k. The decryption is computed as following:

$$\begin{aligned} Dec(c, k, \eta) &= c + F_k^\eta \\ &= c + H_k^\eta - H_k^{(\eta-1)} \\ &= m \end{aligned}$$

The '+' and '–' operations in the above computations are group addition and subtraction operations, respectively.

Note that, we handle the decimal numbers in our encryption/decryption computations in a way similar to how we handle integers, i.e., without any further encoding, due to such lightweight operations whereas Paillier cryptosystem requires exponentiation operations and encoding of decimal numbers.

4.3.2 Additive Homomorphism

Let c_1 and c_2 be two ciphertexts of the plaintext values m_1 and m_2. The addition of these two ciphertexts in the encrypted domain results in:

$$\begin{aligned} c_1 + c_2 &= m_1 + m_2 - F_k^{\eta_1} - F_k^{\eta_2} \\ &= \mathcal{C}_\mathcal{A} \end{aligned}$$

Decryption of $\mathcal{C}_{\mathcal{A}}$ is computed as follows:

$$\begin{aligned} Dec(\mathcal{C}_{\mathcal{A}}, k, \eta_1, \eta_2) &= \mathcal{C}_{\mathcal{A}} + F_k^{\eta_1} + F_k^{\eta_2} \\ &= m_1 + m_2 \end{aligned}$$

The subtraction operation outcome $c_1 - c_2 = \mathcal{C}_{\mathcal{S}}$ in the encrypted domain can also be decrypted as following:

$$\begin{aligned} Dec(\mathcal{C}_{\mathcal{S}}, k, \eta_1, \eta_2) &= \mathcal{C}_{\mathcal{S}} + F_k^{\eta_1} - F_k^{\eta_2} \\ &= m_1 - m_2 \end{aligned}$$

4.3.3 Aggregation

Let $c_1, c_2, \cdots, c_j$ be a set of j ciphertexts where for some η value the nonces used are $\eta_1 = \eta + 1$, $\eta_2 = \eta + 2$, $\cdots$, $\eta_j = \eta + j$, respectively, and we need to compute the aggregation value. Adding all these ciphertexts results in-

$$\begin{aligned} c_1 + \ldots + c_j = m_1 &+ H_k^{\eta} - H_k^{(\eta+1)} \\ &+ m_2 + H_k^{(\eta+1)} - H_k^{(\eta+2)} + \ldots \\ &+ m_j + H_k^{(\eta+j-1)} - H_k^{(\eta+j)} \\ &= m_1 + m_2 + \ldots + m_j + H_k^{\eta} - H_k^{(\eta+j)} \end{aligned}$$

Since η and j values are known to the decryption end, the aggregation (i.e., $m_1 + m_2 + \ldots + m_j$) can be simply decrypted as $c_1 + c_2 + \ldots + c_j - (H_k^{\eta} - H_k^{(\eta+j)})$. Therefore, summation in the encrypted domain is as simple as summation in the plaintext domain. Moreover, decryption of the sum of a large number of encrypted data leverages a telescoping series

and requires only two invocations of the cryptographic hash function H which makes the aggregation in the encrypted domain a very light-weight operation.

4.3.4 Security Analysis

The security analysis is based on a game between the adversary Adv (constrained to run in polynomial time) and the challenger Chl. As many times as the Adv would like, it can select two plaintext messages of its own choosing and provide them to the Chl for encryption where Chl would return a ciphertext encrypting only one of the messages. The advantage of Adv is determined by its probability of guessing the message that is encrypted by Chl.

The adversary Adv selects two plaintext messages m_0 and m_1 and given the encryption function Enc, the Chl computes $(m_b - F_k^{\eta_b})$ where bit $b \in \{0, 1\}$ is chosen randomly by Chl. Along with m_0 and m_1, Adv could also choose the nonce values (η_0 and η_1) itself; however, it is restricted from choosing the same nonce value more than once (*note that we always choose a* ***unique nonce for every ciphertext*** *while utilizing* `Trident` *in our privacy-preserving anomaly detection framework*). After obtaining $(m_b - F_k^{\eta_b})$, Adv outputs a bit b' to guess which message the Chl has chosen to encrypt. Adv wins the game if $b' = b$ and vice versa.

Note that the encryption scheme is semantically secure if for adversary Adv, $Pr[b = b'] < \frac{1}{2} + v(n)$, where $v(n)$ is some negligible function. On the other hand, Adv can guess the correct bit with probability greater than $\frac{1}{2} + v(n)$ only if it can guess $F_k^{\eta_b}$. Since $H_k^{\eta_b}$ and $H_k^{\eta_b - 1}$ are generated using a cryptographically secure pseudo-random number generator, $H_k^{\eta_b} - H_k^{(\eta_b - 1)} = F_k^{\eta_b}$ is also pseudo-random, and thus cannot be guessed by the Adv. Therefore, $Pr[b = b'] < \frac{1}{2} + v(n)$ holds for the adversary Adv which proves the encryption scheme to be semantically secure. □

4.3.5 Malleability

Note that, our `Trident` encryption scheme is malleable, i.e., it is possible for the adversary to transform a ciphertext into another ciphertext which would decrypt to a related plaintext. To prevent the adversary from exploiting this vulnerability, we can use integrity protection techniques such as Message Authentication Code (MAC) to guard against any tampering of the ciphertext.

4.4 Privacy-preserving Anomaly Detection Framework

In this section, we describe our privacy-preserving anomaly detection framework for point, contextual, and collective anomalies in Sections 4.4.1, 4.4.2, and 4.4.3, respectively.

4.4.1 Privacy-preserving Point Anomaly Detection

Algorithm 4 performs point anomaly detection without providing any data privacy, i.e., the ESP observes all the data points in plaintext including the μ and σ values for each step size. The privacy-preserving version of Algorithm 4 is given in Algorithm 5 (steps 1-2 from Algorithm 4 are omitted to avoid redundancy). In this version, the sensor devices from the DO end send the data points in an encrypted form to the ESP. Note that the ESP only receives $E_i.c$ (i.e., the ESP does not receive $E_i.\eta$) since the computations at the ESP end are independent of the nonce values. Furthermore, the DO end can choose any initial value for the nonce and increment it sequentially. For the purpose of clarity, the initial nonce value is set to 1 in Algorithm 5. Since the edge receives the data in encrypted form, it cannot perform arbitrary computations on the data and therefore some of the intermediate computations for anomaly detection are performed by the device (computations at the DO end are shown in blue text in Algorithm 5). Note that, the computations at the DO end are infrequent (i.e., only when there is a requirement to recompute the model) and thus neither result in significant computation latency (discussed in Section 4.5) nor significant energy consumption.

1: **for** i = 0,. . ., $m-1$ (m data points) **do**
2: DO : collects d_i
3: DO : $z = \frac{(d_i-\mu)}{\sigma}$
4: DO : $AnomScore_i$ = 1 - $\frac{1}{2}\ erfc(\frac{z}{\sqrt{2}})$
5: DO : $E_i = Enc(d_i, k, i+1)$
6: DO : sends $E_i.c$ and $AnomScore_i$ to the ESP
7: **if** $i > 0$ **then**
8: $enc\_sum[i] = enc\_sum[i-1]$ + $E_i.c$
9: **else**
10: $enc\_sum[i] = E_i.c$
11: **end if**
12: **if** $i > 0$ & $i\%stepSize = 0$ & $i \leq windowSize$ **then**
13: ESP sends $enc\_sum[i]$ to DO
14: DO : $sum[i] = Dec(enc\_sum[i], i+1, k)$
15: DO : $\mu = \frac{sum[i]}{(i+1)}$
16: DO : $E_\mu = Enc(\mu, k, r+1)$
17: DO : sends $E_\mu.c$ to the ESP
18: ESP sends $E_k.c - E_\mu.c$ to DO for $k = 0, \ldots, i$
19: DO : σ = $\sqrt{\frac{\sum_{k=0}^{i} Dec(E_k.c-E_\mu.c)^2}{(i+1)}}$
20: **end if**
21: **if** $i > 0$ & $i\%stepSize = 0$ & $i > windowSize$ **then**
22: ESP sends $enc\_sum[i]$, $enc\_sum[i - windowSize]$ to DO
23: DO : $sum[i] = Dec(enc\_sum[i], i+1, k)$
24: DO : $sum[i - windowSize] = Dec(enc\_sum\ [i - windowSize], i - windowSize + 1, k)$
25: DO : $\mu = \frac{sum[i]-sum[i-windowSize]}{windowSize}$
26: DO : $E_\mu = Enc(\mu, k, r+1)$
27: DO : sends $E_\mu.c$ to the ESP
28: ESP sends $E_k.c - E_\mu.c$ to DO for $k = i - windowSize, \ldots, i$
29: DO : $\sigma = \sqrt{\frac{\sum_{k=i-windowSize}^{i} Dec(E_k.c-E_\mu.c)^2}{windowSize}}$
30: **end if**
31: Append $E_i.c$ to encrypted dataset $E_D = \{E_1, E_2, \ldots, E_{i-1}\}$ along with its corresponding $AnomScore_i$ (for storage)
32: **end for**

Algorithm 5: Privacy-preserving Algorithm for windowed Gaussian anomaly detector.

The DO end always possesses the current μ and σ values in the plaintext form so that it can compute the anomaly score of a collected data point d_i instantly using the Q-function in equation 4.5 (where $z = \frac{d_i - \mu}{\sigma}$). The next step for the DO is to encrypt d_i using the `Trident` encryption scheme as described in Section 4.3.1. Finally, the DO sends the encrypted d_i along with the corresponding anomaly score to the ESP. The ESP also computes an array of summations of received encrypted values (i.e., $enc\_sum[i] = enc\_sum[i-1] + E_i.c$). Note that `Trident` is additively homomorphic, and therefore the encrypted d_i values can be aggregated in the encrypted domain. If there is a requirement of recomputing the μ and σ values (depending on the value of the $stepSize$), the ESP provides the sum value of the data points in the current window to the DO. After computing the μ, the DO sends this updated μ in encrypted form to the ESP. The nonce for encrypting μ is randomly chosen from a previously defined range that does not overlap with the nonces used for encrypting data values. Also, the same nonce is never used for encrypting two μ values. The ESP then returns the array of differences between the encrypted window data points and the encrypted mean μ (the communication cost for this step can be decreased by considering a smaller sample from the window data points and computing the differences from μ only for the data points in the sample, as discussed later in Section 4.5). Finally, the DO computes σ and is ready to compute the anomaly score of the next data point with updated μ and σ values.

Figure 4.2 presents the privacy-preserving anomaly detection results on *realAdExchange* data. The second plot shows the encrypted data that is sent to the ESP and used by the ESP for the computations given in Algorithm 5. The third plot shows the anomalous points detected by the algorithm (i.e., the points with anomaly score 1 and all other points' anomaly scores are omitted). *Note that both Algorithms 4 and 5 detect the same set of anomalies, i.e., there is no compromise with respect to accuracy when the privacy-preserving algorithm (Algorithm 5) is used to detect the anomalies.*

The symmetric key (used for both encryption and decryption operations) is never exposed to the ESP. Only the DO possesses this key to encrypt the data before off-loading

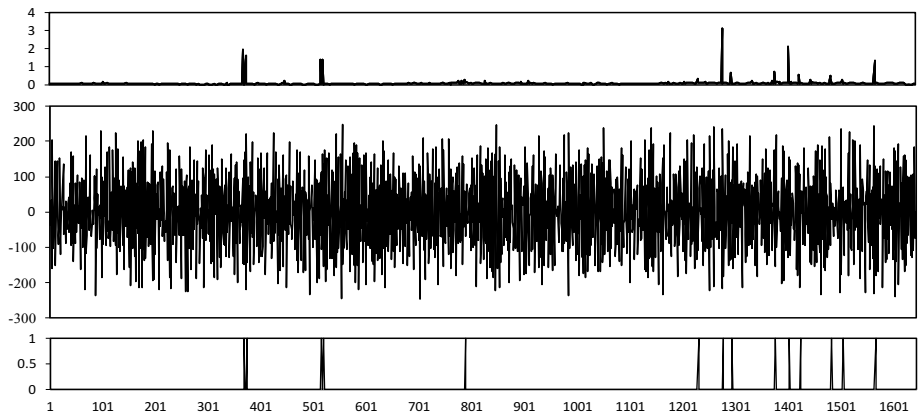


Figure 4.2.: The plots show the original data, encrypted data, and the detected anomalies using Algorithm 5, respectively.

to the ESP and to decrypt some intermediate values for recomputing the parameters μ and σ when needed.

4.4.2 Privacy-preserving Contextual Anomaly Detection

In many applications, two or more data streams could be related to each other. For instance, a sensor that measures energy consumption of electrical boxes may determine that a particular box is consuming an anomalously high amount of energy. However, when this anomalous event is viewed in context with the humidity/temperature data collected by another sensor, the high energy consumption may not be anomalous at all. In such cases, we can compute the anomaly scores in the same way as Section 4.4.1 on different data streams individually (for example, one for energy consumption data and another for humidity/temperature data). Finally, the ESP can determine if there is any contextual anomaly by monitoring and comparing these anomaly scores while the privacy of both data streams (energy and humidity) is preserved by the `Trident` encryption. Note that the rules to define contextual anomalies highly depend on the nature of the application and thus need to be set by the DO and communicated to the ESP.

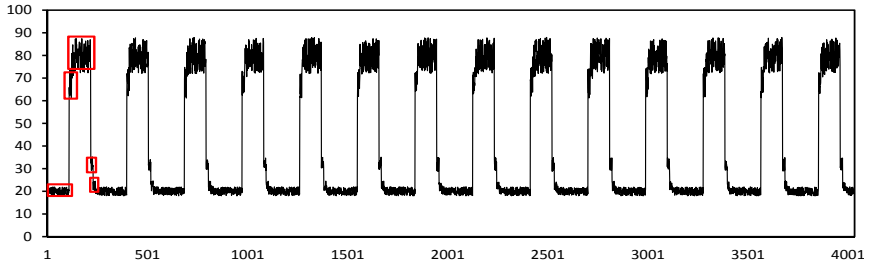
4.4.3 Privacy-preserving Collective Anomaly Detection

Collective anomalies refer to the cases where the individual data instances may not be anomalous by themselves but their occurrence together as a collection is anomalous with respect to the entire data set [116]. Though in point anomaly cases the data trend is mostly consistent in normal situations (with insignificant variation from one data point to the next as shown in Figure 4.1(b)) and one Gaussian distribution is sufficient, in the case of collective anomalies, multiple Gaussian distributions are more appropriate, e.g., see Figure 4.3(a) plotted from [117]. Evidently, there are five data models, namely D^1, D^2, D^3, D^4, and D^5, that interleave by alternating among each other with the average values 20.02, 67.88, 79.53, 31.71, and 22.4, respectively, as marked in Figure 4.3(a). The numbers of data points that belong to these models are 2184, 168, 1344, 168, and 168, respectively.

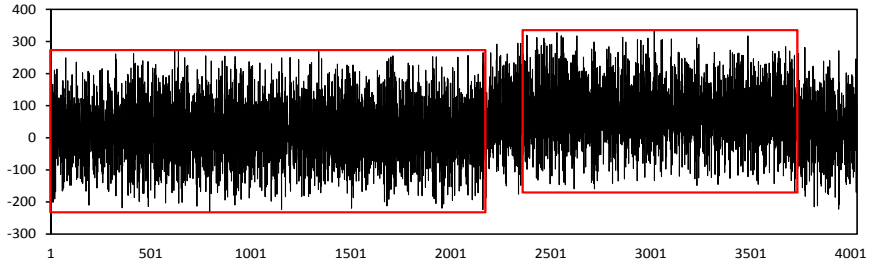
If a single Gaussian distribution is used for modeling this dataset, the μ value would be ~ 42.44 while the σ value (28.08) would be unnecessarily large which would result in misclassification of benign and anomalous data points. For example, if there is a spike of value 80 within the first data model marked in Figure 4.3(a), i.e., when the expected value is ~ 20, the anomaly score of that data point would not be 1 since only a single Gaussian distribution is considered which accepts all data points in the range $[20, 80]$ as normal.

Therefore, it is evident that complex datasets need to be modeled with multiple Gaussian distributions. In the following, we discuss different possible solutions, their limitations, and finally, propose a complete solution able to detect the collective anomalies without compromising privacy.

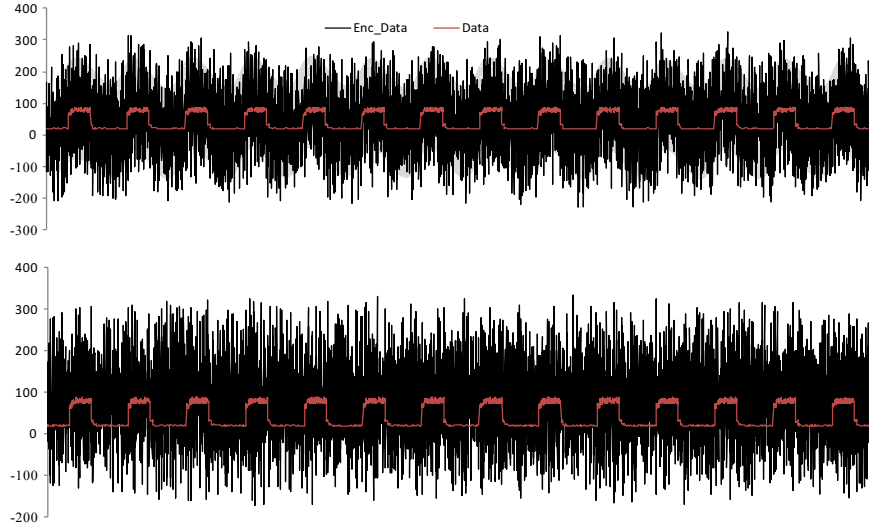
• **Independent multiple Gaussian distributions:** A simple solution could be to use completely separate Gaussian distributions with non-shared η values, i.e., each Gaussian distribution with its own η series. For instance, the dataset plotted in Figure 4.3(a) would have 5 Gaussian distributions and their starting η values η_1^1, η_1^2, η_1^3, η_1^4, and η_1^5 all would be initialized to 1. However, as discussed in Section 4.3.4, the nonce η chosen for each ciphertext has to be unique in order to ensure that the encryption scheme is semantically secure. Gaussian distributions using *far away* initial points, e.g., η_1^1 = 1, η_1^2 = 100,000, η_1^3



(a)



(b)



(c)

Figure 4.3.: (a) A complex dataset, (b) Plot of the encrypted data points for the datasets D^1, D^2, D^3, D^4, and D^5 consecutively with independent Gaussian distributions under unique keys, (c) First plot shows encrypted data without offsetting which reflects the underlying plaintext data whereas the encrypted data with offsetting in the second plot removes such underlying pattern.

= 200,000, etc. might seem to be a possible solution. However, in the cases where data are generated with very high velocities, such solution would become impractical and thus fail. Therefore, the idea of independent multiple Gaussian distributions is neither practical nor secure.

• **Multiple Gaussian distributions with unique keys:** Since it is not secure to use the same nonce twice under the same key, another possible solution is to use unique keys for these independent Gaussian distributions. For instance, the dataset plotted in Figure 4.3(a) could have five unique keys k^1, k^2, k^3, k^4, and k^5 for the five data models, that is, D^1, D^2, D^3, D^4, and D^5, respectively, while their starting η values η_1^1, η_1^2, η_1^3, η_1^4, and η_1^5 all could be initialized to 1 without losing semantic security. However, we have observed that there is a clear distinction among the encrypted data values for the five data models (e.g., encrypted data points from D^3 have larger values than the encrypted data points from D^1 which reflects the pattern of the plaintext data). Given the dissimilarity between the data points belonging to D^1 and D^3, using a different key for these models further increases the difference and thus compromises the privacy of the plaintext data. In order to demonstrate the contrast, we extract the encrypted data points belonging to each data model and plot them consecutively in the order D^1, D^2, D^3, D^4, and D^5, as shown in Figure 4.3(b). Note that the ESP is able to do the same since it requires to know the data model boundaries to determine the corresponding windows (see Algorithm 5). The two rectangles marked in Figure 4.3(b) represent the encrypted data points belonging to D^1 and D^3 (2184 and 1344 data points, respectively). From the plot, it is evident that the encrypted data points do not look completely random, and when these data models interleave, the encrypted data points reveal the pattern of the underlying plaintext data to the ESP. This is due to the fact that though `Trident` is **not** a strictly order preserving encryption scheme, it collectively preserves the pattern in the encrypted domain. Figure 4.3(b) also concludes that *simply using an order preserving encryption scheme to encrypt the data points cannot be a reliable solution*.

• **Multiple Gaussian distributions with encrypted data offsetting and interleaving η values**: First, in order to reduce the dissimilarity among interleaving data models we: (1)

use a single key and (2) offset the encrypted data values of the different data models so that the distributions of all data models are similar. Second, to ensure semantic security, the different data models share the η values (i.e., only one sequence of η values interleave among multiple data models).

For instance, the mean of the D^1 data values is 20.02 whereas the mean of the D^3 data values is 79.53. In order to ensure indistinguishable distribution of these data models, we offset all the encrypted data values of the D^1 dataset with the value $79.53 - 20.02 = 59.51$. Similarly, the encrypted data values of the D^2, D^4, and D^5 datasets are also increased according to their respective differences with the highest valued data model (which is D^3 for this example).

To compute the sum of the encrypted data values of the current window, the ESP needs to know the boundaries of the data models. For the ease of understanding, let's assume that the original data series contains two distinct data models D^1 and D^2, where each period of the data contains 4 data points belonging to each data model. If the window size is set to 8, the aggregation of the first 8 points of D^1 (i.e., $c_1 + \ldots + c_4 + c_9 + \ldots + c_{12}$) would be computed as follows:

$$\begin{aligned} & m_1 + H_k^0 - H_k^1 + \ldots + m_4 + H_k^3 - H_k^4 \\ & + m_9 + H_k^4 - H_k^9 + \ldots + m_{12} + H_k^{11} - H_k^{12} \end{aligned}$$

which results in $m_1 + \ldots + m_4 + m_9 + \ldots + m_{12} + H_k^0 - H_k^{12}$, and requires the addition of $(H_k^{12} - H_k^0)$ for decryption. Note that, the encryption of m_9 is $m_9 + H_k^4 - H_k^9$ instead of $m_9 + H_k^8 - H_k^9$ to maintain the consistency between two appearances of data points from the same data model D^1.

After the sum of a window data values is computed by the ESP and is returned to the DO, the DO first removes the offset values from the sum (the offset values are pre-computed from the historical data, e.g., the offset value for data model D^1 is 59.51) and then decrypts the sum in order to compute the μ. The rest of the steps are similar to that of Algorithm 5. Figure 4.3(c) shows the plots of the encrypted data points before and after

offsetting, respectively. Whereas the first plot reveals the underlying pattern of the plaintext data (demonstrated by a grey shade at the background), the encrypted data in the second plot removes such pattern with the idea of offsetting.

Although we use offsetting to ensure indistinguishable distribution, extensive differences in the standard deviations among different data models may leak the information that one data model has higher standard deviation than the other. However, a larger ciphertext domain can minimize such differences in the distribution since it would reduce the standard deviation gap between the data models. Therefore, by combining offsetting with a large enough ciphertext domain, we avoid any such information leakage of the plaintext data, even in the cases of extensive differences in the standard deviations.

4.4.4 Privacy-preserving Anomaly Detection for More Complicated Scenarios

Graph-based anomaly detection: Our proposed framework can be adapted for more complicated anomaly detection scenarios, e.g., dynamic (i.e., time evolving) graph anomaly detection algorithms that aim to identify anomalous patterns using a moving window analysis [118, 119]. For instance, let G be a graph representing communications among $n = |V(G)|$ entities (people in a social network, computers in a network, etc.) and an edge (b, d) representing the communication activities (e.g., number of messages exchanged) between the entities represented by vertices b and d. Graph snapshots that have unusually high communication activities compared to the past or that exceed a pre-defined threshold could signal anomalous events in the graph. Our framework can detect such anomalous events in a privacy-preserving manner where the communication activity data among entities are encrypted with `Trident` scheme and any anomalous activity between two entities is detected using Algorithm 5 as described in Section 4.4.1. Also, such privacy-preserving graph-based anomaly detection can be extended to identify local sub-regions in a graph with excessive activities. Note that with respect to scalability, this method would scale linearly with number of edges in the graph.

Malicious ESP: We consider a single, honest but curious (passive adversary model) ESP in our solution. Though rare (so that the business reputation is not hampered), the ESP could also be malicious and may arbitrarily deviate from the protocol which would disrupt the correctness of the protocol. This active adversary model is out of the scope of this work. However, practical verifiable computation techniques [120] can be leveraged where the ESP, along with the results of the computations for which the ESP is responsible, would append a proof that the computation has been carried out correctly.

Table 4.2.: Anomaly detection scenarios.

Scenario	Edge	Computation	Privacy
TCN	**T**rusted	**C**entralized	**N**one
UDH	**U**ntrusted	**D**istributed	**H**igh

4.5 Evaluation

In this section, we first introduce our experiment setup in Section 4.5.1. Then in Section 4.5.2, we report the results of a comparison between the `Trident` and Paillier encryption schemes. In the subsequent subsections, we perform three sets of analysis: (i) In Subsection 4.5.3, we evaluate the accuracy of our privacy-preserving anomaly detection framework for different anomaly cases. (ii) In Subsection 4.5.4, we evaluate the overhead of our privacy-preserving scenario, *UDH*, where the edge is untrusted and the computation steps for anomaly detection are distributed over DO and ESP, and compare it with that of the baseline scenario, *TCN*, where the device transfers its entire raw data to the edge, i.e., there is no privacy of the data (refer to Table 4.2). (iii) In Subsection 4.5.5, we perform a scalability analysis to evaluate the overhead of *UDH* scenario with increasing dataset sizes and compare its overhead with that of the baseline scenario *TCN*. **Note that, since the solution for privacy-preserving collective anomaly detection subsumes the solutions for point and contextual anomaly detection, we focus on the datasets with only collective anomalies for the experiments in this section.**

4.5.1 Experiment Setup

For testbed experiments, we have used a Raspberry Pi 3 (CPU: 4 ARM Cortex-A53, 1.2GHz, RAM: 1GB LPDDR2 900 MHz, Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless) to represent the device at the DO end whereas the ESP end is represented by an Intel(R) Core(TM) i7-3770 CPU machine (3.4 GHz Dual-Core, Ram: 8GB).

We implemented our solution in Python v2.7.3, where we used ZeroMQ [121] for communication between the device and the edge. We present the simulation results on datasets from Numenta Anomaly Benchmark (NAB) [122] and UCI Machine Learning Repository [108].

$$\mu = \frac{1}{N}\sum_{i=1}^{10}\Big(\sum_{j=1}^{N/10} d_j^i\Big),\ \sigma = \Big(\frac{1}{N-1}\sum_{i=1}^{10}\Big(\sum_{j=1}^{N/10}(d_j^i-\mu)^2\Big)\Big)^{1/2}$$

Computing μ and σ: In order to learn these parameters initially, we use a training dataset of N data points that does not contain any anomalous points (i.e., manually verified before the anomaly detection system deployment). We use 10-fold cross validation, i.e., the training dataset is divided into 10 equal-sized non-overlapping subsets and there are 10 training executions each time using a new subset for validation and the rest 9 as training datasets (as shown in the above equations). The μ and σ values with the lowest error among the 10 resultant models are then used as the final parameter values. In the case of multiple data models (e.g., the dataset in Figure 4.3(a)), the training phase recognizes the abrupt changes between data models and computes these parameters for each data model individually.

4.5.2 Comparison Between the Trident and Paillier Schemes

To compare the `Trident` scheme with the Paillier cryptosystem (which requires expensive exponentiation operations), we encrypt the data points in the dataset used in Figure 4.3(a). We vary the number of data points and compute the latency for encryption of these data points. Figure 4.4 shows the latency difference between the two encryption

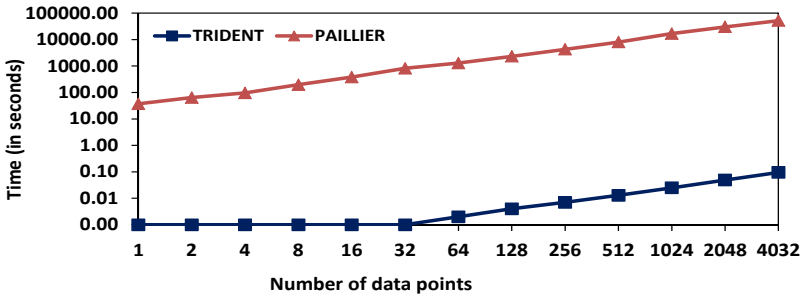


Figure 4.4.: Latency comparison between `Trident` and Paillier.

schemes as the number of data points increases. For instance, the latency for Paillier encryption with 4032 data points is $\sim$ 51801556 milliseconds whereas for our `Trident` encryption scheme the latency is only $\sim$ 95 milliseconds. Moreover, the summation of 4032 encrypted data points takes only 12 milliseconds for our encryption scheme whereas for the Paillier encryption scheme the latency is $\sim$ 400 milliseconds.

4.5.3 Performance Analysis

To evaluate the performance of our privacy-preserving anomaly detection algorithm in detecting anomalies in the data, we consider four anomaly cases [123]: *flat-middle*, *jumps-down*, *jumps-up*, and *no-jump*. Figure 4.5 plots these anomaly cases along with their anomaly scores in Figure 4.5(a), 4.5(b), 4.5(c), and 4.5(d), respectively. These results are obtained by setting $windowSize$ = 200, $stepSize$ = 50, and $sampleSize$ = 50 (details on $sampleSize$ is provided in Section 4.5.4). Considering an anomaly threshold of 0.99, i.e., considering any data point with anomaly score $>$ 0.99 as anomalous, the above setting results in an accuracy of 99.97% with 99.17% precision and 100% recall for the above four anomaly cases. Even in the cases of anomalous data points spanning over multiple data models (e.g., in the *flat-middle* anomaly case, the anomalous data points span over all the five data models), our privacy-preserving framework could identify the anomalies in the test dataset with this high accuracy.

Varying window size: To understand the correlation between the window size and the resultant accuracy, we vary the window size of our anomaly detection algorithm. Fig-

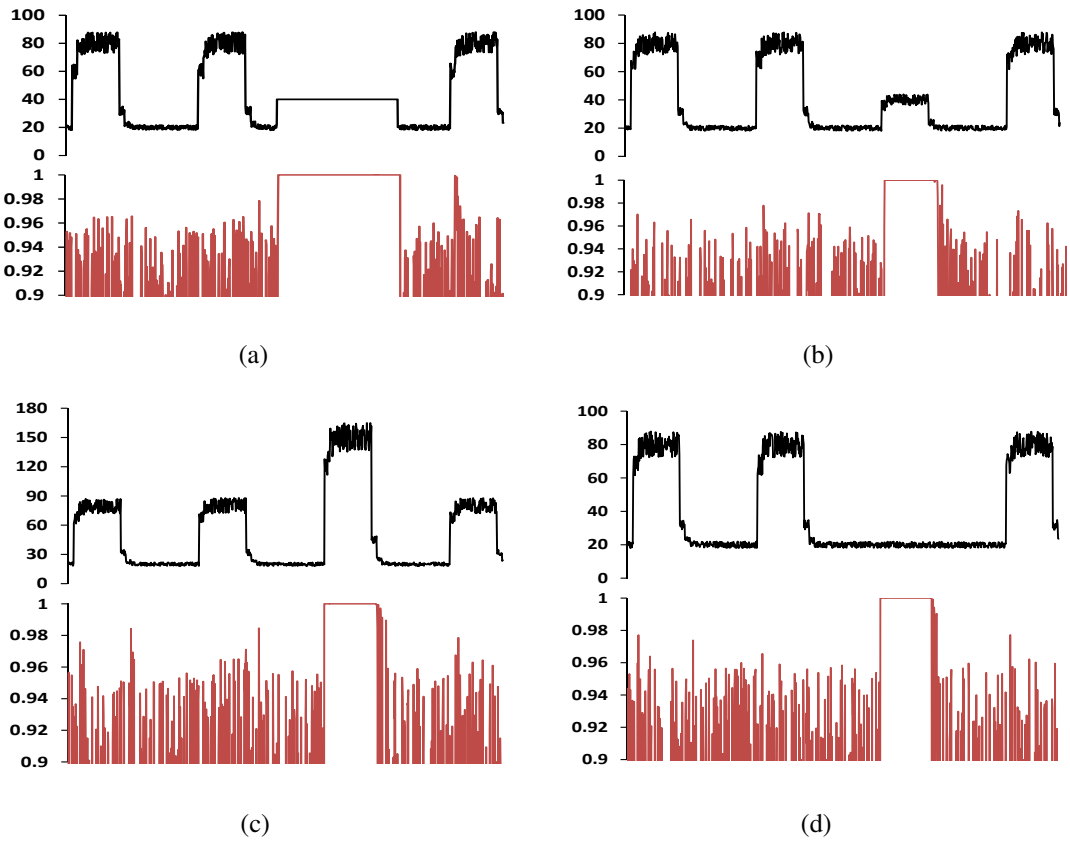
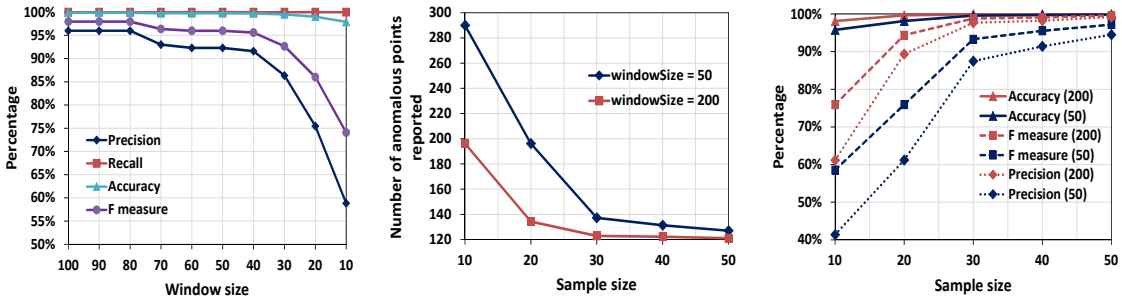


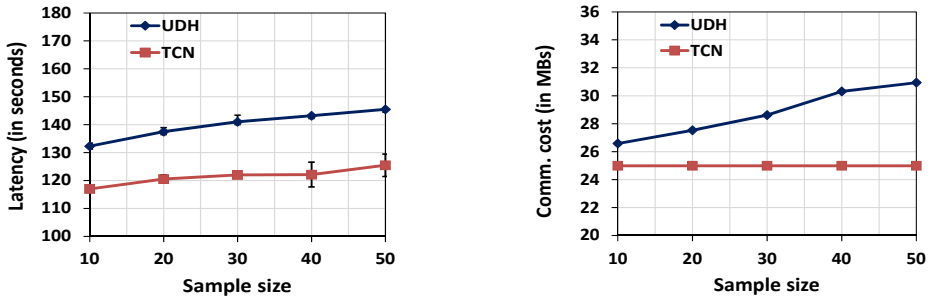
Figure 4.5.: Anomaly detection performance for the cases: (a) *flat-middle*, (b) *jumps-down*, (c) *jumps-up*, and (d) *no-jump*. The plots in black color represent the plaintext data, the plots in red represent the corresponding anomaly scores.

ure 4.6(a) presents the precision, recall, accuracy, and f-measure for ten different window sizes $\{100, 90, \ldots, 10\}$ while setting the step size to 10. As it can be seen in the figure, the performance of anomaly detection degrades with the smaller window sizes. This is due to the fact that the models learned from a small window size (e.g., 10) are not generalized enough to classify the future data points with high accuracy. On the other hand, as described in Algorithm 5, a larger window size results in higher communication overhead since the ESP sends to the DO an array of size $windowSize$ each time the model parameter σ is updated. In order to address this trade-off, in the next section, we introduce the idea of sampling the window data points which results in comparable accuracy while minimizing the communication overhead significantly.



(a) (b) (c)

Figure 4.6.: Impact on anomaly detection performance with varying window sizes and varying sample sizes.



(a) (b)

Figure 4.7.: Comparison between TCN and UDH in terms of latency and communication cost for different sample sizes.

4.5.4 Overhead Analysis

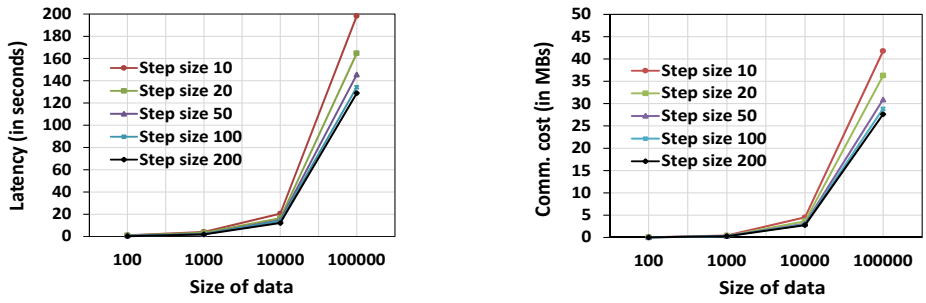
Sampling while computing σ: While updating the σ value of the Gaussian distribution after every step size, the ESP needs to send to the DO the array of encrypted differences between the current μ and the current window data points. Therefore, a larger window size results in higher overhead both in terms of latency and communication cost. However, this additional overhead can be reduced by t times if instead of sending an array of the window size, a smaller sample from the window data points is considered and only the differences of those data points from the current μ value are sent to the DO. For instance, if the window

size is 200 and only 20 data values are sampled randomly from the window, it would result in 10 times less overhead.

Since sampling the window data points could result in significantly improved latency and communication cost, in order to understand the performance differences between *small window size* and *large window size with sampling*, we compare the two window sizes 200 and 50 for the *jumps-down* data, while for each of these cases we sample $10, 20, 30, 40, 50$ data points from the window. Figure 4.6(b) shows the numbers of anomalous points reported for window sizes 200 and 50 with varying sample sizes. The *jumps-down* data originally has 120 anomalies (note that the recall for both window sizes is 100%). From the results it is evident that considering a larger window size results in less false positives, even if the sample size is the same. For instance, with a sample size 30 (i.e., we randomly sample 30 points from the window data points independent of the window size), the case with $windowSize$ = 50 results in 17.2 false positives whereas the case with $windowSize$ = 200 results in only 2.9 false positives in average (the average is computed by running the experiments 10 times each). Figure 4.6(c) compares the accuracy, precision and f-measure (we exclude recall since the recall rate is the same) of these two window sizes for the sample sizes $10, 20, 30, 40, 50$. Since setting $windowSize$ = 200 and $sampleSize$ = 50 result in a negligible false positive rate (as shown in Figure 4.6(b)), we chose these parameter values in the experiments described in Section 4.5.3 and Figure 4.5.

Comparison between *TCN* and *UDH* scenarios: In order to further assess the overhead of our privacy-preserving framework, we implemented both Algorithms 4 and 5. Algorithm 4 represents the *TCN* scenario where the DO off-loads its raw data to the edge and outsources the task of anomaly detection as a whole (refer to Table 4.2). In contrast, Algorithm 5 represents the *UDH* scenario.

In Figure 4.7, we show the overhead comparison between the *TCN* and *UDH* scenarios in terms of latency and communication cost for the synthetic dataset consisting of 100000 data points from the UCI Machine Learning Repository [108]. We vary the sample size to be 50, 40, 30, 20, and 10 whereas the window and step sizes are set to 200 and 50, respectively.



(a) (b)

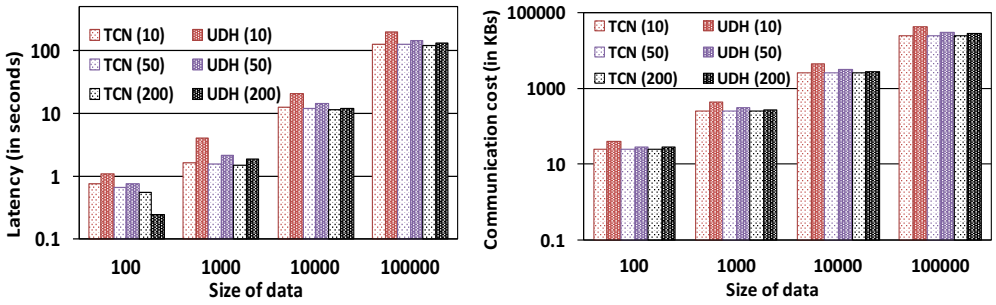
Figure 4.8.: Scalability analysis for UDH in terms of latency and communication cost for varying data and step sizes.

As shown in Figure 4.7(a), the latency for the *UDH* scenario increases as the sample size increases (the corresponding performances in anomaly detection are given in Figure 4.6(c)). This is due to the fact that for increasing values of the sample size, the ESP samples more data in each model update step and thus results in higher communication latency. For the *TCN* scenario, the increase rate in the latency for larger sample sizes is slower than that of *UDH*. This is because in this case the ESP and DO do not need to communicate since the ESP already has the data in plaintext. The slight increase in the latency for the *TCN* scenario is due to sampling more data at the ESP end for each model update procedure. The results for the *UDH* scenario in Figure 4.7(b) are consistent with the results in Figure 4.7(a). However, for the *TCN* scenario, the communication cost in Figure 4.7(b) does not change for different step sizes since for this scenario the communication cost is incurred only when the DO sends the raw data to the ESP and, therefore, the communication cost is not impacted by the anomaly detection task at all. In this experiment, i.e., for anomaly detection on 100000 data points with $windowSize$ = 200, $stepSize$ = 50, and $sampleSize$ = 50, our *UDH* scenario, while preserving the privacy of the data, has a latency overhead of 15.9% and a communication cost overhead of 23.81% over the *TCN* scenario whereas the *TCN* scenario does not consider the privacy at all. Note that, depending on the nature of the data, the communication overhead could be as low as 6.3% with $sampleSize$ = 10 as shown in Figure 4.7(b).

4.5.5 Scalability Analysis

In order to perform the scalability analysis of our privacy-preserving anomaly detection algorithm, we sample 100, 1000, and 10000 data points from the original 100000 points dataset to generate datasets of different sizes. Moreover, since the $stepSize$ requirement for different application scenarios could be very different (i.e., the expected frequency of updating the model) and since the value of $stepSize$ could affect the latency and communication overhead (i.e., smaller $stepSize$ would result in higher overhead), for the experiments in this section, we vary the step size to be 10, 20, 50, 100, and 200 whereas the window size is fixed at 200 and the sampling size is fixed at 50. We then compute the overheads for the *UDH* scenario as shown in Figure 4.8. As the size of the data increases, the latency for anomaly detection naturally increases as shown in Figure 4.8(a). Moreover, the latency for this *UDH* scenario increases as the step size decreases. This is due to the fact that for decreasing values of the step size, the Gaussian parameters are recomputed more frequently which increases the communication between the ESP and DO and thus results in higher latency. For example, the latency for privacy-preserving anomaly detection on 100000 data points are $\sim$198.3, $\sim$164.6, $\sim$145.5, $\sim$134.2, and $\sim$128.8 seconds for the step sizes 10, 20, 50, 100, and 200, respectively. In this experiment, *we do not explicitly consider the data arrival rate* of the real-time data collected by some sensor, but compute the amount of time required by our privacy-preserving framework to analyze a certain amount of data and identify any anomalies. Here, for instance, our privacy-preserving framework can analyze 100000 data points in $\sim$145.5 seconds when the step size is set to 50. Note that the latency for the *TCN* scenario with the identical setting is 125.5 seconds.

The communication costs for different data sizes are shown in Figure 4.8(b) which complies with Figure 4.8(a). The communication cost incurred by our privacy-preserving anomaly detection framework on 100000 data points are $\sim$ 41.7, $\sim$ 36.3, $\sim$ 30.9, $\sim$ 28.8, and $\sim$ 27.6 MBs for the step sizes 10, 20, 50, 100, and 200, respectively. Note that the communication cost to send the entire raw data to the ESP is $\sim$ 24.9 MBs, which is the communication overhead for the baseline *TCN* scenario.



(a) (b)

Figure 4.9.: Comparison between TCN and UDH in terms of latency and communication cost for varying data and step sizes.

A detailed overhead comparison between *TCN* and *UDH* scenarios on scalability is given in Figure 4.9. Figure 4.9(a) compares the latency of the *TCN* and *UDH* scenarios for different data sizes and also for varying step sizes, that is, 10, 50, and 200. The latency difference between these two scenarios increases as the step size decreases. This is due to the fact that for the *TCN* scenario, the increase rate in the latency for smaller step sizes is slower than that of *UDH*. This is because in this case the ESP and DO do not need to communicate since the ESP already has the data in plaintext. The slight increase in the latency for the *TCN* scenario is due to more frequent Gaussian parameters' re-computation at the ESP side. For instance, for the dataset with 100000 data points, the latency difference for step size 200 is only 10.68 seconds whereas the latency difference for step size 50 is 20.02 seconds. Again, this is due to the fact that in the *UDH* scenario, smaller step sizes result in more communication between the ESP and DO for more frequent Gaussian parameters' re-computation. Figure 4.9(b) shows the communication cost comparison between *TCN* and *UDH* scenarios which also reflects the results in Figure 4.9(a). Note that, for the *TCN* scenario, the communication cost does not change for different step sizes since for this scenario the communication cost is incurred only when the DO sends the raw data to the ESP and, therefore, the communication cost is not impacted by the anomaly detection task. Therefore, the communication cost difference between these two scenarios increases as the step size decreases. For instance, for the dataset with 100000 data points, the com-

munication overhead difference for step size 200 is only 2.63 MBs whereas the difference for step size 50 is 5.92 MBs. Note that depending on the nature of the data this overhead could be reduced drastically even for small step sizes by further reducing the sample size which is set at 50 for these experiments.

In summary, though the *UDH* scenario has higher overhead than the *TCN* scenario for very small step sizes, the *TCN* scenario does not provide any data privacy. Moreover, in many applications, such as in industrial data, recomputing the Gaussian parameters every 10 data points may not be required. With larger step sizes, the overhead difference between the two scenarios become insignificant whereas only the *UDH* scenario is privacy-preserving.

4.5.6 Privacy Analysis

According to the techniques described in Section 4.4.3 (i.e., encrypted data offsetting and interleaving η), the only knowledge the ESP can obtain from the encrypted data and anomaly scores is the boundaries of different data models. However, it cannot learn which data model has larger values (since it can only observe the encrypted data values after offsetting) and thus cannot learn the pattern of the underlying plaintext data. Moreover, the boundaries of different data models can be **obscured** by breaking down the consecutive data points belonging to a single data model into multiple unequal parts. For example, if there are 100 consecutive data points from a single data model, the DO could ask for the sums of 30, 50, and 20 data points separately and add the 3 sums to compute the final sum of the 100 data points (this would mislead the ESP to infer that there are data model boundaries after 30, 50, and 20 data points). Also, in our framework, after computing the anomaly scores, the DO end sends these values to the ESP in plaintext for storage. However, if the **privacy of the anomaly scores** is a matter of concern (i.e., the DO does not want the ESP to know if there is an anomaly), the DO can itself store the anomaly scores. Due to the storage constraints at the DO end, the DO may store only the indexes

of the anomalous data points along with their anomaly scores. The storage overhead would be negligible considering anomalies to be rare events.

To understand why our sequential (or, correlated) nonces preserve semantic security, it is important to notice how we use nonces during encryption. We first concatenate the nonce with the key which results in $(k||n)$ and then we compute a cryptographic hash function on this concatenation, i.e., $H(k||n)$. Later, in the security analysis, we prove that $H(k||n)$ - $H(k||n-1)$ = $F(n,k)$ is pseudo-random. The cryptographic hash functions and then computing $F(n,k)$ provides a layer of randomization on the nonces. Again, we depend on the secrecy of the key only, and without the key k, the attacker cannot compute $H()$ or $F()$ even if the nonces are correlated.

4.6 Related Work

Although there exist approaches for anomaly detection in IoT [124, 125], none of these consider privacy. Emami-Naeini et. al [126] design a methodology to identify privacy expectations in the IoT world. In what follows, we provide an overview of existing approaches for privacy-preserving analytics broadly classified into five categories.

Homomorphic Encryption (HE)-based strategies: Techniques in this class have the ability to perform computations directly on encrypted data and therefore require fewer interactions between the ESP and DO while removing the requirement of a trusted third party. While our `Trident` scheme belongs to this category as a partial homomorphic encryption (PHE) scheme, there also exist fully homomorphic encryption (FHE) schemes [127] that support arbitrary computations on encrypted data with the trade-off that these schemes are too complex to be used in practical applications. Several schemes [128, 129] have been recently proposed that focus on reducing the complexity of the original FHE scheme. However, unfortunately, such schemes still remain impractical in terms of latency and communication overhead for use in large-scale and/or real-time applications. There are many other additively homomorphic encryption schemes, e.g., [130], [131], [132], [133], [134], and [135]. Some PHE schemes [131] may support addition and multiplication simultane-

ously but unfortunately are not secure enough [136]. Some recent work aim to leverage partial homomorphic encryption schemes for different real-life applications [137].

Anonymization-based strategies: These approaches partition attributes in a given database into two sets − those containing Personally Identifiable Information (PII) and the rest that do not, and remove the set of PII (e.g., 'Name', 'Social Security Number'). Attributes such as "Zip code", "Age", "Gender", etc. which can identify an individual when used in combination are anonymized using techniques such as k-anonymity [138], l-diversity, etc. However, anonymization based strategies often degrade the utility of the data for anomaly detection [139].

Differential privacy-based strategies: Existing work that leverage differential privacy either assume a co-operative environment for anomaly detection [27] or utilize crowd-sourcing [28]. These assumptions do not directly apply to our scenario where there is a single data owner willing to outsource the anomaly detection task while simultaneously preserving the privacy of the data. Moreover, differential privacy based solutions would require injecting a significant amount of fake data to preserve the privacy of the actual data which in turn would reduce the utility significantly.

SMC-based strategies: These strategies use cryptographic protocols such as Yao's garbled circuits [23], secret sharing [140], etc. Most SMC-based strategies rely on peer-to-peer communication and are usually defined in 2-party scenarios, with extension to multi-party scenarios often resulting in significant communication overhead. Moreover, some SMC-based techniques are exposed to several security vulnerabilities from different weak assumptions, e.g., the assumption of a trusted third party [141].

Randomization-based strategies: These approaches are based on randomization techniques, such as additive data perturbation and random subspace projection [142]. While these approaches are fast and efficient, they do not provide strong security guarantees and are often susceptible to attacks [26].

5. CONCLUSION AND FUTURE WORK

The simplest solution to defend against the ransomware is using backups. Though this is trivial, it would not be a practical assumption when there are millions of people using machines for personal and organizational purposes while being connected to the Internet. Also even when the backup is in place, users may lose changes that are executed after the most recent backup. This raises the requirement of very frequent backup procedure which makes the assumption even more impractical. Therefore, given the size of today's ransomware threat, it is necessary that early ransomware detection systems be installed for the protection of personal and organization data. In the first part of this thesis, we introduce `RWGuard` that detects crypto-ransomware on a user's machine in real-time while removing the false positives due to the user's benign file operations. We evaluate `RWGuard` against 14 most prevalent ransomware families. Our experiments show that `RWGuard` is effective in early detection of ransomware with only negligible false positives ($\sim$0.1%) and zero false negatives while incurring an overhead of only $\sim$1.9%. Furthermore, `RWGuard` recovers all files that are encrypted using CryptoAPI by the corresponding ransomware.

In Ghostbuster, we have proposed a technique to create fine-grained profiles of file system users and to use these profiles for detecting anomalous accesses to file systems. We consider that these anomalous accesses are due to the abuse of data by an insider or by an external attacker who can gain access to the files by exploiting the vulnerabilities of the software or by stealing credentials using different techniques, e.g., man-in-the-middle attack, key-logging, phishing, and so on. However, we learn the normal access patterns of the users by utilizing the block level access information from the OS kernel space and detect such malicious accesses with an accuracy of 98.7%. Note that our AD mechanism can be integrated with any existing anomaly response system [143] which would take actions automatically when our AD mechanism detects an anomaly. Examples of such responses include: sending an alarm to the system/security administrator, disconnecting the user, or

blocking the user from accessing the file. Also, note that a malicious security administrator may abuse the profiles and execute different attacks including masquerading and data harvesting. For protection against such scenarios, we consider the system to have multiple administrators with separation of duty policy [144]. Furthermore, our AD mechanism can be used to monitor accesses to the files storing the profiles (e.g., accesses by system administrators).

In order to address the cases of anomaly detection that require outsourcing the analysis of privacy sensitive data (e.g., health data, financial data), we design an edge computing-based model that detects point, contextual, and collective anomalies in streaming large scale sensor data while preserving the privacy of the data simultaneously. We consider the windowed Gaussian anomaly detector algorithm for detecting anomalies and propose a privacy-preserving version of this algorithm that performs anomaly detection while the edge service provider only sees the encrypted data. In our solution, the sensitive data is never stored in plaintext. While the storage task is outsourced to the edge service provider, the anomaly detection task is distributed between the edge service provider and the device. We build a lightweight and aggregation optimized encryption scheme, `Trident`, which results in practical latency and communication overhead without compromising the accuracy of the anomaly detection results. Finally, we evaluate the performance of the proposed solution in terms of accuracy of the resulting model, communication cost, and latency, and compare it with a baseline scenario where the data is off-loaded to the edge service provider in plaintext and the edge service provider performs anomaly detection on plaintext data. We also perform a scalability analysis to determine the performance of our approach as the size of the data is increased.

Future Work

Effective Anomaly Detection Systems Against a Variety of Advanced Attacks: Despite having various security measures deployed to counter cyber attacks such as data breaches, we still have a long way to go before we can, as a whole, catch up with the grow-

ing threats of cyber crimes. While cyber attacks are becoming increasingly sophisticated and dangerous, we, as security enthusiasts, aspire to develop effective anomaly detection systems against a powerful class of attackers that aim to compromise data confidentiality, integrity, and availability. For instance, in order to protect against advanced ransomware, we plan to profile the existing encryption libraries and in real-time scan the process's memory for similar operations so that we can recover the keys used for encryption and restore the files. Moreover, we plan to take snapshots of the ransomware processes' memories before terminating the processes and analyze those for traces of encryption/decryption keys. There are also novel attacks (e.g., [145]) that demonstrate that it is possible to avoid the generation of significant behavioral features and evade a ransomware detector by distributing the overall malware workload across a small set of cooperating processes. Also, there could be advanced insider threats, such as, a group of insiders evading the detection by remaining within the set threshold individually while still stealing sensitive data in a combined fashion. Though such collusion attacks are less popular among insiders, this is an intelligent way of exfiltrating sensitive information and therefore, we aim to identify these challenging attacks in the future. Moreover, handling concept drifts in benign activities over time when there is no role-based access control system and also, handling adversarial concept drifts are also part of our future work.

Practical Real-Time Machine Learning over Encrypted Data: Many emerging domains, such as autonomous systems and Industrial Internet of Things (IIoT) require real-time machine learning to support automated and effective decision making. *Velocity* and *volume* are two of the most salient features of the data generated by these emerging domains, and thus must be taken care of while processing the data. While many companies (e.g., Google, Microsoft) provide real-time analytics services, unfortunately, these services are not compatible with applications where the data requiring such real-time analytics are privacy-sensitive, e.g., personal healthcare systems and critical industrial monitoring systems. This is due to the shortfall that machine learning algorithms are designed to execute on plaintext data where the privacy issues take a back seat even if the data used for training and decision making are sensitive. Moreover, despite the privacy promises by third-party

service providers, due to the recent cyber attacks and massive data breaches, many individuals/organizations are understandably hesitant to share their data in plaintext with third-parties. Therefore, our work on privacy-preserving real-time anomaly detection has several possible extensions in terms of *scalability*, *dealing with multiple parties*, and last but not the least *incorporating other real-time machine learning requirements*– as discussed in the following:

(i) In large-scale systems such as the Industrial Internet of Things (IIoT) with thousands of nodes deployed, the task of monitoring for anomalous trends indicative of a pending system failure becomes reasonably complex,

(ii) This challenge is further complicated when the data is shared among multiple parties who have a common interest in learning models and making real-time decisions from the union of their data, e.g., multiple autonomous vehicles guiding themselves to drive without human intervention, and

(iii) Besides anomaly detection and predictive maintenance, there are other applications such as real-time e-commerce recommendation systems where the individuals are concerned about their data privacy and may not want to share their usage behaviors on e-commerce sites (e.g., dating sites) in plaintext with the service providers.

In response to these emerging privacy issues, we plan to develop practical real-time machine learning solutions over *encrypted* data that would simultaneously preserve the privacy of such sensitive data.

Evaluating Privacy and Security Assurance of Machine Learning Models: While a variety of companies are streamlining their business processes by adopting machine learning technologies and leveraging commercial ML-as-a-service APIs, it is equally important to understand if these technologies are introducing attack vectors against the privacy of the data on which the models were trained [146, 147]. This is because, in many cases, the datasets used for training such models are proprietary and confidential. Part of my current research [148] addresses and evaluates the vulnerabilities of model inversion attacks on commercially available ML-as-a-service APIs. With the recent advancements in adversarial machine learning, we also plan to evaluate the security assurance of machine learning

models that specifically have security applications, e.g., if an adversary with access to only a black-box model can determine whether a target data instance (an individual, an event, etc.) has been used for training the model (also known as membership inference attack). The consequence of such attacks can be far more devastating when the machine learning model has security applications, such as machine learning for intrusion detection in cyber-physical systems. For instance, an adversary might aim to learn the malicious events an intrusion detection system *is able* to detect. By querying the model of this intrusion detection system multiple times, the adversary might be able to learn those malicious events– if the model is vulnerable to membership inference attack. This may eventually help the adversary to design a sophisticated attack input (i.e., a malicious event) which the victim model would classify as benign. Moreover, poisoning attacks, where the adversary is able to inject malicious training data and manipulate the output of the trained models, have been shown to be effective against anomaly detection models [149, 150]. These kinds of attacks simply undermine the foundations of such models with security applications and thus need to be investigated urgently. With our experience in intrusion detection systems, we aim to rigorously evaluate the vulnerability of such attacks and build theoretical foundations to decide if a particular model is vulnerable. Designing solutions that would eliminate such privacy and security vulnerability from the models while also conserving the models' utility is also a part of our future work.

REFERENCES

REFERENCES

[1] T. Fox-Brewster, "Petya or notpetya: Why the latest ransomware is deadlier than wannacry." FORBES, June 2017. [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2017/06/27/petya-notpetya-ransomware-is-more-powerful-than-wannacry

[2] J. C. Wong and O. Solon, "Massive ransomware cyber-attack hits nearly 100 countries around the world." Theguardian, May. [Online]. Available: https://www.theguardian.com/technology/2017/may/12/global-cyber-attack-ransomware-nsa-uk-nhs

[3] H. Van Riper, "Return of the worm: A red hat analysis." Digital Shadows, September 2017. [Online]. Available: https://www.digitalshadows.com/blog-and-research/return-of-the-worm-a-red-hat-analysis/

[4] E. Bertino, *Data Protection from Insider Threats*, ser. Synthesis Lectures on Data Management. San Rafael: Morgan & Claypool Publishers, 2012.

[5] Google, https://marketingplatform.google.com/about/analytics.

[6] Microsoft, https://azure.microsoft.com/en-us/solutions/architecture/anomaly-detection-in-real-time-data-streams.

[7] Anomaly, https://anomaly.io/.

[8] "Always encrypted," *https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine*, October 2019.

[9] M. I. Sarfraz, M. Nabeel, J. Cao, and E. Bertino, "Dbmask: Fine-grained access control on encrypted relational databases," *Trans. Data Privacy*, vol. 9, no. 3, p. 187–214, Dec. 2016.

[10] R. Poddar, T. Boelter, and R. A. Popa, "Arx: An encrypted database using semantically secure encryption," *Proc. VLDB Endow.*, vol. 12, no. 11, p. 1664–1678, Jul. 2019. [Online]. Available: https://doi.org/10.14778/3342263.3342641

[11] J. Lee, J. Lee, and J. Hong, "How to make efficient decoy files for ransomware detection?" in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. New York, NY, USA: ACM, 2017, pp. 208–212.

[12] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148*, ser. DIMVA 2015. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 3–24.

[13] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "Unveil: A large-scale, automated approach to detecting ransomware," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 757–772.

[14] A. Kharraz and E. Kirda, "Redemption: Real-time protection against ransomware at end-hosts," in *Research in Attacks, Intrusions, and Defenses*, 2017, pp. 98–119.

[15] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, "Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 2231–2244.

[16] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "Cryptolock (and drop it): Stopping ransomware attacks on user data," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, June 2016, pp. 303–312.

[17] "Security breach at sony– here's what you need to know," *http://www.forbes.com/sites/josephsteinberg/2014/12/11/massive-security-breach-at-sony-heres-what-you-need -to-know/*, December 2014.

[18] D. M. Andrew P. Moore, Michael Hanley, "A pattern for increased monitoring for intellectual property theft by departing insiders," Carnegie Mellon University, Tech. Rep., 2012, http://www.sei.cmu.edu/reports/12tr008.pdf.

[19] A. P. Moore, M. L. Collins, D. A. Mundie, R. M. Ruefle, and D. M. McIntire, "Pattern-based design of insider threat programs," Carnegie Mellon University, Tech. Rep., 2014, http://resources.sei.cmu.edu/asset_files/technicalnote/2014_004_001_427430.pdf.

[20] A. Kamra, E. Terzi, and E. Bertino, "Detecting anomalous access patterns in relational databases," *The VLDB Journal*, vol. 17, no. 5, pp. 1063–1077, Aug. 2008.

[21] S. R. Hussain, A. M. Sallam, and E. Bertino, "Detanom: Detecting anomalous database transactions by insiders," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '15, 2015, pp. 25–35.

[22] B. Juba, C. Musco, F. Long, S. Sidiroglou-douskos, and M. Rinard, "Principled sampling for anomaly detection," in *Proceedings of the Network and Distributed System Security Symposium*, 2015.

[23] A. C.-C. Yao, "How to generate and exchange secrets," *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, vol. 00, pp. 162–167, 1986.

[24] X. Yi, A. Bouguettaya, D. Georgakopoulos, A. Song, and J. Willemson, "Privacy protection for wireless medical sensor data," *IEEE Trans. on Dependable and Secure Computing*, vol. 13(3), pp. 369–380, May 2016.

[25] K. Bhaduri, M. D. Stefanski, and A. N. Srivastava, "Privacy-preserving outlier detection through random nonlinear data distortion," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 1, pp. 260–272, Feb 2011.

[26] Z. Huang, W. Du, and B. Chen, "Deriving private information from randomized data," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 37–48.

[27] J. Vaidya and C. Clifton, "Privacy-preserving outlier detection," in *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*, Nov 2004, pp. 233–240.

[28] M. Maruseac, G. Ghinita, B. Avci, G. Trajcevski, and P. Scheuermann, "Privacy-preserving detection of anomalous phenomena in crowdsourced environmental sensing," in *Advances in Spatial and Temporal Databases*. Cham: Springer International Publishing, 2015, pp. 313–332.

[29] D. C. Montgomery, G. C. Runger, and N. F. Hubele, "Engineering statistics." Wiley, 2006.

[30] A. Jayanthi, "First known ransomware attack in 1989 also targeted healthcare." Becker's Hospital Review, May 2016. [Online]. Available: http://www.beckershospitalreview.com/healthcare-information-technology/first-known-ransomware-attack-in-1989-also-targeted-healthcare.html

[31] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "Shieldfs: A self-healing, ransomware-aware filesystem," in *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: ACM, 2016, pp. 336–347.

[32] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, "Paybreak: Defense against cryptographic ransomware," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS. New York, NY, USA: ACM, 2017, pp. 599–611.

[33] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, "Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection," *ArXiv e-prints*, Sep. 2016.

[34] Kryptel. [Online]. Available: https://www.kryptel.com/products/kryptel.php

[35] M. Inc., "File system minifilter drivers," May 2014. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/hardware/ff540402(v=vs.85).aspx

[36] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[37] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[38] V. Roussev, *Data Fingerprinting with Similarity Digests*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 207–226.

[39] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Trans. Inf. Theor.*, vol. 37, no. 1, pp. 145–151, Sep. 2006.

[40] VirusTotal. [Online]. Available: https://www.virustotal.com

[41] O. Malware. [Online]. Available: http://openmalware.org

[42] VXVault. [Online]. Available: http://vxvault.siri-urz.net/URL_List.php

[43] Zelster. [Online]. Available: https://zeltser.com/malware-sample-sources/

[44] Malc0de. [Online]. Available: http://malc0de.com/rss

[45] D. Xu, J. Ming, and D. Wu, "Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping," in *Proceedings 2017 IEEE Symposium on Security and Privacy*, May 2017, pp. 129–140.

[46] J. K. Lee, S. Y. Moon, and J. H. Park, "Cloudrps: a cloud analysis based enhanced ransomware prevention system," *The Journal of Supercomputing*, vol. 73, no. 7, pp. 3065–3084, Jul 2017.

[47] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics," *CoRR*, vol. abs/1611.08294, 2016.

[48] N. Andronio, S. Zanero, and F. Maggi, "Heldroid: Dissecting and detecting mobile ransomware," in *Proceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 9404*, ser. RAID. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 382–404.

[49] D. Y. Huang and et al., "Tracking ransomware end-to-end," in *Proceedings of the 2018 IEEE Conference on Security and Privacy*, ser. SP'18, 2018.

[50] C. V. Bijitha, R. Sukumaran, and H. V. Nath, "A survey on ransomware detection techniques," in *Secure Knowledge Management In Artificial Intelligence Era*, S. K. Sahay, N. Goel, V. Patil, and M. Jadliwala, Eds. Singapore: Springer Singapore, 2020, pp. 55–68.

[51] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents." Springer Berlin Heidelberg, 2009, pp. 51–70.

[52] CryptoStopper. [Online]. Available: www.watchpointdata.com/cryptostopper/

[53] J. Calvet, J. M. Fernandez, and J.-Y. Marion, "Aligot: Cryptographic function identification in obfuscated binary programs," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2012, pp. 169–182.

[54] P. Lestringant, F. Guihéry, and P.-A. Fouque, "Automated identification of cryptographic primitives in binary code with data flow graph isomorphism," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS. New York, NY, USA: ACM, 2015, pp. 203–214.

[55] A. Sallam, E. Bertino, S. R. S. Hussain, D. Landers, M. Lefter, and D. Steiner, "Db-safe—an anomaly detection system to protect databases from exfiltration attempts," *IEEE Systems Journal*, vol. 11, no. 2, pp. 483–493, 2017.

[56] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[57] J. Park and R. Sandhu, "Originator control in usage control," in *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, 2002, pp. 60–66.

[58] E. Bertino and G. Ghinita, "Towards mechanisms for detection and prevention of data exfiltration by insiders: Keynote talk paper," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '11, 2011, pp. 10–19.

[59] "Cybersecurity watch survey: How bad is the insider threat?" Carnegie Mellon University, Tech. Rep., 2012, http://resources.sei.cmu.edu/asset_files/Presentation/2013_017_101_57766.pdf.

[60] C. Huth and R. Ruefle, "Components and considerations in building an insider threat program," Carnegie Mellon University, Tech. Rep., 2013, http://resources.sei.cmu.edu/asset_files/Webinar/ 2013_018_101_69083.pdf.

[61] M. Collins, D. M. Cappelli, T. Caron, R. F. Trzeciak, and A. P. Moore, "Spotlight on: Programmers as malicious insiders (updated and revised)," Carnegie Mellon University, Tech. Rep., 2013, http://resources.sei.cmu.edu/asset_files/WhitePaper/2013_019_001_85232.pdf.

[62] E. Bertino, A. Kamra, and J. P. Early, "Profiling database application to detect sql injection attacks," in *IEEE International Performance, Computing, and Communications Conference, IPCCC 2007*, April 2007, pp. 449–458.

[63] S. Mathew, M. Petropoulos, H. Q. Ngo, and S. Upadhyaya, "A data-centric approach to insider attack detection in database systems," in *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*, ser. RAID'10, 2010, pp. 382–401.

[64] D. Fadolalkarim and E. Bertino, "A-pandde: Advanced provenance-based anomaly detection of data exfiltration," *Computers Security*, vol. 84, pp. 276 – 287, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404819300823

[65] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, pp. 18 – 28, 2009.

[66] M. V. Mahoney and P. K. Chan, "Learning nonstationary models of normal network traffic for detecting novel attacks," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02, 2002, pp. 376–385.

[67] M. Thottan and C. Ji, "Anomaly detection in ip networks," *Signal Processing, IEEE Transactions on*, vol. 51, no. 8, pp. 2191–2204, Aug 2003.

[68] "Detecting insider information theft using features from file access logs," in *Computer Security - ESORICS 2014*, ser. Lecture Notes in Computer Science, vol. 8713, 2014.

[69] "Anomaly detection in computer security and an application to file system accesses," in *Foundations of Intelligent Systems*, ser. Lecture Notes in Computer Science, M.-S. Hacid, N. Murray, Z. Raś, and S. Tsumoto, Eds., vol. 3488, 2005.

[70] L. Huang and K. Wong, "Anomaly detection by monitoring filesystem activities," in *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ser. ICPC '11, 2011, pp. 221–222.

[71] *ZFS End-to-End Data Integrity*, https://blogs.oracle.com/bonwick/entry/zfs_end_to_end_data.

[72] S. Patil, A. Kashyap, G. Sivathanu, and E. Zadok, "Fs: An in-kernel integrity checker and intrusion detection file system," in *Proceedings of the 18th USENIX Conference on System Administration*, ser. LISA '04, 2004, pp. 67–78.

[73] M. B. Salem, S. Hershkop, and S. J. Stolfo, *A Survey of Insider Attack Detection Research*. Boston, MA: Springer US, 2008, pp. 69–90.

[74] D. E. Denning, "An intrusion-detection model," *IEEE Trans. Softw. Eng.*, vol. 13, no. 2, pp. 222–232, Feb. 1987.

[75] S. Mehnaz and E. Bertino, "Ghostbuster: A fine-grained approach for anomaly detection in file system accesses," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ser. CODASPY '17, 2017, pp. 3–14.

[76] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences," *Data Min. Knowl. Discov.*, vol. 1, no. 3, pp. 259–289, Jan. 1997.

[77] D. Fadolalkarim, A. Sallam, and E. Bertino, "Pandde: Provenance-based anomaly detection of data exfiltration," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '16, 2016, pp. 267–276.

[78] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB, 1994, pp. 487–499.

[79] S. Laxman, P. S. Sastry, and K. P. Unnikrishnan, "A fast algorithm for finding frequent episodes in event streams," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07, 2007, pp. 410–419.

[80] J. Vaidya, V. Atluri, and J. Warner, "Roleminer: Mining roles using subset enumeration," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS, 2006, pp. 144–153.

[81] J. Vaidya, V. Atluri, and Q. Guo, "The role mining problem: Finding a minimal descriptive set of roles," in *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '07, 2007, pp. 175–184.

[82] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo, "Mining roles with semantic meanings," in *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '08, 2008, pp. 21–30.

[83] G. Cleuziou, "An extended version of the k-means method for overlapping clustering," in *2008 19th International Conference on Pattern Recognition*, Dec 2008, pp. 1–4.

[84] J. J. Whang, I. S. Dhillon, and D. F. Gleich, "Non-exhaustive, overlapping k-means," in *Proceedings of the 2015 SIAM International Conference on Data Mining*, pp. 936–944.

[85] H. Takabi and J. B. Joshi, "Stateminer: An efficient similarity-based approach for optimal mining of role hierarchy," in *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '10, 2010, pp. 55–64.

[86] S. Shetty, S. K. Mukkavilli, and L. H. Keel, "An integrated machine learning and control theoretic model for mining concept-drifting data streams," in *2011 IEEE International Conference on Technologies for Homeland Security (HST)*, Nov 2011, pp. 75–80.

[87] "Handling adversarial concept drift in streaming data," *Expert Systems with Applications*, vol. 97, pp. 18 – 40, 2018.

[88] *Intel® Software Guard Extensions (Intel® SGX)*, https://software.intel.com/sgx/.

[89] H. Mazzawi, G. Dalal, D. Rozenblatz, L. Ein-Dorx, M. Niniox, and O. Lavi, "Anomaly detection in large databases using behavioral patterning," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, April 2017, pp. 1140–1149.

[90] *Block I/O Layer Tracing using blktrace*, http://smackerelofopinion.blogspot.com/2009/10/block-io-layer-tracing-using-blktrace.html.

[91] "Towards a taxonomy of intrusion-detection systems," *Comput. Netw.*, vol. 31, no. 9, pp. 805–822, Apr. 1999.

[92] T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, Q. Chen, and R. A. Bridges, "A survey of intrusion detection systems leveraging host data," *CoRR*, vol. abs/1805.06070, 2018.

[93] D. Dimitrios, M. S. A., K. Georgios, P. Maria, C. Nathan, and G. Stefanos, "Evaluation of anomaly-based ids for mobile devices using machine learning classifiers," *Security and Communication Networks*, vol. 5, no. 1, pp. 3–14.

[94] N. Vrakas and C. Lambrinoudakis, "An intrusion detection and prevention system for ims and voip services," *International Journal of Information Security*, vol. 12, no. 3, pp. 201–217, Jun 2013.

[95] H. T. Nguyen, K. Franke, and S. Petrovic, "Towards a generic feature-selection measure for intrusion detection," in *Proceedings of the 2010 20th International Conference on Pattern Recognition*, ser. ICPR '10, 2010, pp. 1529–1532.

[96] N. Baracaldo and J. Joshi, "A trust-and-risk aware rbac framework: Tackling insider threat," in *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, 2012, pp. 167–176.

[97] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward supervised anomaly detection," *J. Artif. Int. Res.*, vol. 46, no. 1, pp. 235–262, Jan. 2013.

[98] Y. Li, N. Wu, S. Wang, and S. Jajodia, "Enhancing profiles for anomaly detection using time granularities," *J. Comput. Secur.*, vol. 10, no. 1-2, pp. 137–157, Jul. 2002.

[99] S. Mehnaz and E. Bertino, "Building robust temporal user profiles for anomaly detection in file system accesses," in *14th Annual Conference on Privacy, Security and Trust (PST)*, 2016, pp. 207–210.

[100] S. S. Srivastava, M. Atre, S. Sharma, R. Gupta, and S. K. Shukla, "Verity: Blockchains to detect insider attacks in dbms," 2019.

[101] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, *Baiting Inside Attackers Using Decoy Documents*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 51–70.

[102] B. M. Bowen, V. P. Kemerlis, P. Prabhu, A. D. Keromytis, and S. J. Stolfo, "Automating the injection of believable decoys to detect snooping," in *Proceedings of the Third ACM Conference on Wireless Network Security*, ser. WiSec '10, 2010, pp. 81–86.

[103] I. Ray and N. Poolsapassit, "Using attack trees to identify malicious attacks from authorized insiders," in *Proceedings of the 10th European Conference on Research in Computer Security*, ser. ESORICS'05, 2005, pp. 231–246.

[104] T. E. e. a. Senator.

[105] W. Claycomb, D. Shin, and G.-J. Ahn, "Enhancing directory virtualization to detect insider activity," *Security and Communication Networks*, vol. 5, no. 8.

[106] J. B. Camiña, J. Rodríguez, and R. Monroy, *Towards a Masquerade Detection System Based on User's Tasks*. Cham: Springer International Publishing, 2014, pp. 447–465.

[107] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, May 2010.

[108] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml.

[109] "Iot attacks are getting worse and no one's listening." CNET.

[110] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the internet of things: perspectives and challenges," *Wireless Networks*, vol. 20, no. 8, pp. 2481–2501, Nov 2014.

[111] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[112] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, "Data security and privacy-preserving in edge computing paradigm: Survey and open issues," *IEEE Access*, vol. 6, pp. 18 209–18 237, 2018.

[113] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security and Privacy Magazine*, vol. 8, no. 6, pp. 24–31, Nov 2010.

[114] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *12th IFIP/IEEE International Symposium on Integrated Network Management and Workshops*, May 2011, pp. 385–392.

[115] "Realadexchange data (exchange-4_cpc_results.csv)," https://github.com/numenta/NAB/tree/master/data/realAdExchange.

[116] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, pp. 15:1–15:58, 2009.

[117] "Artificial data with no anomaly (art_daily_small_noise.csv)," https://github.com/numenta/NAB/tree/master/data/artificialNoAnomaly.

[118] C. E. Priebe, J. M. Conroy, D. J. Marchette, and Y. Park, "Scan statistics on enron graphs," *Computational & Mathematical Organization Theory*, vol. 11, no. 3, pp. 229–247, Oct 2005.

[119] M. Mongiovì, P. Bogdanov, R. Ranca, E. E. Papalexakis, C. Faloutsos, and A. K. Singh, "Netspot: Spotting significant anomalous regions on dynamic networks," in *Proceedings of the 2013 SIAM International Conference on Data Mining*, pp. 28–36.

[120] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 238–252.

[121] ZeroMQ, http://zeromq.org/.

[122] Numenta Anomaly Benchmark, https://github.com/numenta/NAB.

[123] "Artificial data with anomaly, note=https://github.com/numenta/nab/tree/master/data/artificialwithanomaly,."

[124] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2661 – 2674, 2013.

[125] D. H. Summerville, K. M. Zach, and Y. Chen, "Ultra-lightweight deep packet anomaly detection for internet of things devices," in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, Dec 2015, pp. 1–8.

[126] P. Emami-Naeini, S. Bhagavatula, H. Habib, M. Degeling, L. Bauer, L. Cranor, and N. Sadeh, "Privacy expectations and preferences in an IoT world," in *SOUPS '17: Proceedings of the 13th Symposium on Usable Privacy and Security*. USENIX, Jul. 2017.

[127] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *In Proc. STOC*, 2009, pp. 169–178.

[128] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. ACM, 2012, pp. 309–325.

[129] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Proceedings of the 31st Annual Conference on Advances in Cryptology*, ser. CRYPTO'11. Springer-Verlag, 2011, pp. 487–504.

[130] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, ser. EUROCRYPT'99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 223–238.

[131] J. Domingo-Ferrer, "A provably secure additive and multiplicative privacy homomorphism," in *Proceedings of the 5th International Conference on Information Security*, ser. ISC '02. London, UK, UK: Springer-Verlag, 2002, pp. 471–483.

[132] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, "Big data analytics over encrypted datasets with seabed," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16, 2016, pp. 587–602.

[133] Ningduo Peng, Guangchun Luo, Ke Qin, and Aiguo Chen, "A fast additively symmetric homomorphic encryption scheme for vector data," in *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, 2013, pp. 2586–2589.

[134] P. G. Carlos Aguilar Melchor and J. Herranz, "Additively homomorphic encryption with d-operand multiplications," Cryptology ePrint Archive, Report 2008/378, 2008, https://eprint.iacr.org/2008/378.

[135] A. C. . Chan, "Symmetric-key homomorphic encryption for encrypted data processing," in *2009 IEEE International Conference on Communications*, 2009, pp. 1–5.

[136] D. Wagner, *Cryptanalysis of an Algebraic Privacy Homomorphism*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 234–239.

[137] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, ser. CCSW '11. New York, NY, USA: ACM, 2011, pp. 113–124.

[138] L. Sweeney, "K-anonymity: A model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, Oct. 2002.

[139] M. Burkhart, D. Brauckhoff, and M. May, "On the utility of anonymized flow traces for anomaly detection," *CoRR*, vol. abs/0810.1655, 2008.

[140] R. L. Rivest, A. Shamir, and Y. Tauman, "How to share a secret," *Communications of the ACM*, vol. 22, no. 22, pp. 612–613, 1979.

[141] S. Yakoubov, V. Gadepally, N. Schear, E. Shen, and A. Yerukhimovich, "A survey of cryptographic approaches to securing big-data analytics in the cloud," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2014, pp. 1–6.

[142] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, "Privacy preserving mining of association rules," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02. New York, NY, USA: ACM, 2002, pp. 217–228.

[143] A. Kamra and E. Bertino, "Design and implementation of an intrusion response system for relational databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 6, pp. 875–888, June 2011.

[144] R. Simon and M. E. Zurko, "Separation of duty in role-based environments," in *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, ser. CSFW '97, 1997, pp. 183–.

[145] F. D. Gaspari, D. Hitaj, G. Pagnotta, L. D. Carli, and L. V. Mancini, "The naked sun: Malicious cooperation between benign-looking processes," 2019.

[146] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 1322–1333. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813677

[147] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 3–18.

[148] S. Mehnaz, N. Li, and E. Bertino, "Evaluating model inversion attacks on machine learning models (under submission)."

[149] R. Bhargava, "Adversarial anomaly detection." PhD dissertation, Purdue University, August 2019.

[150] R. Bhargava and C. Clifton, "Anomaly detection under poisoning attacks," in *Proceedings of the ODD v5.0: Outlier Detection De-constructed Workshop, 24th ACM SIGKDD international conference on Knowledge Discovery and Data Mining (KDD)*, 2018.