future internet

MDPI

Article

Cloud Security Using Fine-Grained Efficient Information Flow Tracking

Fahad Alqahtani [1], Mohammed Almutairi [2,*] and Frederick T. Sheldon [2,*]

[1] Department of Computer Science, Prince Sattam Bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia; fah.alqahtani@psau.edu.sa
[2] Department of Computer Science, University of Idaho, Moscow, ID 83843, USA
* Correspondence: almu9701@vandals.uidaho.edu (M.A.); sheldon@ieee.org or sheldon@uidaho.edu (F.T.S.)

Abstract: This study provides a comprehensive review and comparative analysis of existing Information Flow Tracking (IFT) tools which underscores the imperative for mitigating data leakage in complex cloud systems. Traditional methods impose significant overhead on Cloud Service Providers (CSPs) and management activities, prompting the exploration of alternatives such as IFT. By augmenting consumer data subsets with security tags and deploying a network of monitors, IFT facilitates the detection and prevention of data leaks among cloud tenants. The research here has focused on preventing misuse, such as the exfiltration and/or extrusion of sensitive data in the cloud as well as the role of anonymization. The CloudMonitor framework was envisioned and developed to study and design mechanisms for transparent and efficient IFT (eIFT). The framework enables the experimentation, analysis, and validation of innovative methods for providing greater control to cloud service consumers (CSCs) over their data. Moreover, eIFT enables enhanced visibility to assess data conveyances by third-party services toward avoiding security risks (e.g., data exfiltration). Our implementation and validation of the framework uses both a centralized and dynamic IFT approach to achieve these goals. We measured the balance between dynamism and granularity of the data being tracked versus efficiency. To establish a security and performance baseline for better defense in depth, this work focuses primarily on unique Dynamic IFT tracking capabilities using e.g., Infrastructure as a Service (IaaS). Consumers and service providers can negotiate specific security enforcement standards using our framework. Thus, this study orchestrates and assesses, using a series of real-world experiments, how distinct monitoring capabilities combine to provide a comparatively higher level of security. Input/output performance was evaluated for execution time and resource utilization using several experiments. The results show that the performance is unaffected by the magnitude of the input/output data that is tracked. In other words, as the volume of data increases, we notice that the execution time grows linearly. However, this increase occurs at a rate that is notably slower than what would be anticipated in a strictly proportional relationship. The system achieves an average CPU and memory consumption overhead profile of 8% and 37% while completing less than one second for all of the validation test runs. The results establish a performance efficiency baseline for a better measure and understanding of the cost of preserving confidentiality, integrity, and availability (CIA) for cloud Consumers and Providers (C&P). Consumers can scrutinize the benefits (i.e., security) and tradeoffs (memory usage, bandwidth, CPU usage, and throughput) and the cost of ensuring CIA can be established, monitored, and controlled. This work provides the primary use-cases, formula for enforcing the rules of data isolation, data tracking policy framework, and the basis for managing confidential data flow and data leak prevention using the CloudMonitor framework.

Keywords: cloud; security; cloud security; cloud computing security; data leak prevention; data isolation schema; managing confidential data flow; information flow tracking use-cases

Citation: Alqahtani, F.; Almutairi, M.; Sheldon, F.T. Cloud Security Using Fine-Grained Efficient Information Flow Tracking. *Future Internet* **2024**, *16*, 110. https://doi.org/10.3390/fi16040110

Academic Editor: Antonio Esposito

Received: 31 January 2024
Revised: 9 March 2024
Accepted: 18 March 2024
Published: 25 March 2024

1. Introduction

1.1. Overview

Modern Information Technology (IT) landscapes are transitioning from traditional, capital-intensive systems to utility-based, on-demand models. This shift encompasses various paradigms such as grid computing, cloud computing, and edge computing, offering managed resources to users as services, predominantly delivered over the Internet [1]. In cloud computing, Cloud Service Providers (CSPs) manage resources delivered as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), utilizing multi-tenancy and resource virtualization techniques [2,3]. This model facilitates a user-friendly, on-demand, pay-per-use system, incorporating public, private, and hybrid deployment approaches [4,5]. The integration of cloud computing extends to modern applications like Big Data, the Internet of Things (IoT), Software-Defined Networking (SDN), and Network Function Virtualization (NFV) [6], positioning the cloud at the forefront of today's leading technologies.

With the proliferation of cloud services, ensuring data security and privacy has become paramount. Enterprises, especially those dealing with sensitive information, face challenges in relinquishing control over their data when it resides in the cloud [7]. Concerns arise regarding data security in transit, processing, and storage. Consequently, many enterprises prefer to store and protect sensitive data locally, leading to potential bottlenecks in workflows involving cloud services. Moreover, restrictive security measures can prompt employees to circumvent policies, posing further risks to data integrity, extrusion, and exfiltration.

While some data may be less sensitive and suitable for cloud storage, enterprises often possess highly confidential information that necessitates stringent security measures. In such cases, implementing policies to restrict data propagation beyond internal networks becomes imperative. Information Flow Tracking (IFT) systems play a crucial role in enforcing these policies and preventing data leaks in cloud environments.

1.2. Research Problem

Cloud computing technologies used to supplement IT systems pose real security threats to consumers and providers. Securing data both at the local premises of the consumer and within the cloud services is one of the difficulties that the research community has yet to fully address [8]. Many methods, techniques, and technologies have been proposed to handle these threats. These approaches include: (i) Same Origin Policy (SOP) [9], (ii) Content Security Policy [10], (iii) Cross-Origin Resource Sharing (CORS) [11], and (iv) Sandboxing [12]. These approaches try to mitigate the effect of untrustworthy parties using the same cloud or web service by means of an all-or-nothing approach. The techniques restrict user access to the same service by controlling how access is granted. However, once permission is granted, users have all the privileges over the shared service (i.e., there are no fine-grained controls over user actions). Indisputably, this has created an issue of trust among Cloud Service Consumers (CSC) and has caused some to retreat from cloud usage, while those planning to adopt a cloud service strategy are reluctant.

The conventional security measures embedded within cloud infrastructures, such as authentication, access control, and data privacy mechanisms, are foundational components. However, despite their presence, they often fall short of meeting the diverse and evolving security needs of cloud consumers [13]. These needs encompass various aspects, including stringent data protection requirements, comprehensive access management, and assurance of regulatory compliance. Specifically, conventional security measures struggle to adequately address concerns related to controlling how user information is processed to and from the cloud environment and within. Maintaining trust and ensuring the security of user data in the cloud is exigent. In this context, effective and efficient IFT emerges as a promising approach to address these challenges.

1.3. Motivation

The challenges inherent in cloud services, particularly regarding data confidentiality, integrity, and availability (CIA), drive the need for robust security measures [14]. Organizations utilizing cloud services must maintain control over their data to meet regulatory requirements and protect sensitive information. However, the track record of many Cloud Service Providers (CSPs) in ensuring data security and privacy is inadequate, posing risks to sensitive data [15]. Hence, there is a compelling need for consumer-centered solutions that empower organizations to monitor and control data flows.

1.4. Objectives

This research, based on a comprehensive review of cloud insecurities [16], aims to develop, test, and validate a prototype framework for Information Flow Tracking (IFT) in cloud environments. As such, this work seeks to enhance transparency and control over data propagation, thereby bolstering client trust in the security and privacy of their data. By implementing an efficient and effective IFT, organizations can monitor and enforce policies to prevent sensitive data leaks and mitigate the risk of unauthorized access.

1.5. Research Questions

This study addresses the following questions:

- What appropriate tools and techniques for implementing IFT in cloud environments are established?
- Developing and implementing an IFT-based Cloud Security Framework (CSF), including validation testing?
- How can the effectiveness of the developed "IFT framework" be evaluated to facilitate adoption by both CSCs and CSPs?

1.6. Scope of the Research

This research focused on reviewing various IFT tracking tools and strategies and developing a Consumer Confidential Data (CCD) security measure for testing purposes. We created a simulated enterprise cloud environment, designated as a CSP Testbed we have denoted CloudMonitor, to assess the effectiveness of the developed IFT framework. In doing so, we have:

- Highlighted data vulnerability scenarios within different cloud environments.
- Developed a prototypical secure data framework for safeguarding information.
- Compared the efficiency of existing IFT tools and proposed a framework for securing data in CSC-based and CSP-class cloud environments.

1.7. Limitations of Validation Experiments

The CloudMonitor cannot distinguish between intentional and unintentional sensitive information transmissions because of the transmission gap between cloud storage and user systems. This leads to false alarms and potential widespread issues. Additionally, standard IFT techniques struggle to work seamlessly with older application programs in business settings, which makes it tough to discriminate between intentional and unintentional data transfers. CloudMonitor also faces difficulty in accurately tracking sensitive data propagation because it lacks access to more advanced data structures. The study notes limitations in applying the CloudMonitor framework to various advanced cloud service setups present today, such as those hosted by Big Tech (e.g., Microsoft, Apple, Google, and Amazon), as the assessment mainly focused on traditional open cloud systems. Further research is essential to confirm its effectiveness across different types of cloud services. Nonetheless, this work is based on a comprehensive review of the current literature.

2. Background and Related Work

In this section, an overview of the cloud and its service provisions are given. The threat models in connection with these service models are presented. Subsequently, typical best

practices to secure them are discussed, and an overview of IFT mechanisms and security concerns is provided.

2.1. Cloud Computational Imperatives

Cloud computing, as defined by NIST, is a model that enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. These resources can be rapidly provisioned and released with minimal management effort or service-provider interaction [17].

Main characteristics of cloud computing:

1. On-demand self-service: CSCs can manage, provision, and scale computing power and storage without direct interaction with CSPs [18].
2. Broad network access: Cloud services are accessible via networks, typically through a cloud carrier such as an Internet Service Provider (ISP) or directly via a CSP using specialized internet protocols which constitute the data transmission gap.
3. Resource pooling: CSPs combine storage and computing resources in multi-tenant models, shared among different consumer groups. CSCs have little control over physical resources and processing/resting locations.
4. Rapid elasticity: Cloud services can be rapidly deployed and scaled to match consumer demands, with automatic scaling of resources up/down using the pay-as-you-use model.
5. Measured services: Consumption of cloud resources is optimized, measured, and charged by the CSP's control system.
6. Five essential factors in cloud computing: (i) consumer, (ii) provider, (iii) carrier, (iv) auditor, and (v) broker. The primary stakeholder, the cloud consumer (CSC), engages with a CSP through service agreements, selecting services from the CSP's catalog, and managing payments accordingly.
7. Cloud Monitoring by Major Providers: Companies like Amazon, Microsoft, and Google employ cloud monitoring to track the effectiveness and utilization of their services (e.g., Amazon uses CloudWatch to monitor various components of its infrastructure, enabling quick issue identification and resolution).
8. Three Delivery Provisioning Models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). These models cater to different consumer needs and preferences. See Tables 1 and 2.

Table 1. Delivery provisioning models control and functionality.

Delivery Models	Level of Control Granted to Consumers	Functionality Available to Consumers
SaaS	Utilization and configuration associated with usage.	Access to the front end and user interface.
PaaS	Limited administration	Modest administration-level controlling IT resources to the platform consumers' usage.
IaaS	Full administration	Full access to IT resources linked to virtualized infrastructure, and conceivably access to underlying physical IT resources.

Cloud Deployment Models

There are four deployment models—private cloud, community cloud, public cloud, and hybrid cloud [3]. The cloud, with its infrastructure, is dedicated to a specific organization called a private cloud. The infrastructure can be on- or off-premises. In public clouds, the infrastructure is shared by many mutually untrusted CSCs. The public cloud is usually off-premises. The community cloud is an architectural construct where the infrastructure is accessed by a specific set of CSC organizations of the same interest. The physical imple-

mentation can reside either on or off-premises. The hybrid cloud is a combination of two or more cloud types and is gaining popularity.

Table 2. Consumers Against Provider Activities About Delivery Models.

Delivery Models	Cloud Consumer Activities	Cloud Provider Activities
SaaS	Utilizes and configures cloud	Implements, manages, and maintains cloud services. CSC tracks usage.
PaaS	Develops, tests, deploys, and manages cloud services and cloud-based solutions	Pre-con platforms and allocates the necessary underlying resources, middleware, and other IT resources as needed. The cloud consumer tracks usage.
IaaS	Involves establishing and configuring fundamental infrastructure, and the installation, administration, and monitoring of necessary software.	Offers and controls the physical processing, storage, networking, and hosting required. Cloud consumers track usage.

2.2. CSC Cloud Security Concerns

As consumers increasingly rely on cloud services and data uploads, monitoring for security becomes challenging. Consumer organizations must ensure data security with cloud providers, but providers are often reluctant to share security details, fearing reputational damage [19]. This lack of trust is a major barrier to cloud adoption for sensitive data handling [5]. To address this, a mechanism is needed for consumers to easily assess their data's security in the cloud. It is not necessary for all data to have identical protection standards, and some organizations may opt for cloud services across only certain data sets [14]. Yet, inherent security issues do persist, such as data loss, changing providers, and insider threats.

Requirements for ensuring the security of cloud systems include elements such as authentication, authorization, liability, and privacy [6] and are often expressed collectively as confidentiality, integrity, and availability (CIA). Confidentiality covers data privacy where consumer data is not exposed to third parties without the consumers' expressed consent. Integrity refers to the protection of data stored in the cloud against unauthorized access or modification [15]. Availability refers to the services provided by the CSP being immediately available when expected and needed by the consumer. Authentication is a guarantee that when a person claims ownership of data, it is properly identified and authorized to access the data and consummate services. Authorization assures that a person has the proper access rights and is presumably to act on the designated cloud data in proper accordance [17]. Accountability holds persons or processes accountable for performing an activity on the data (i.e., non-repudiation) [18]. Non-repudiation is complete when the CIA holds and represents the transitive nature of the full repertoire of cloud services.

Therefore, prior to employing the cloud, an organization must be certain about the security properties being provided by the CSP. Cloud services may cause some adoption risks, of which consumers must be made aware. Conversely, CSPs can bring security benefits that may be unknown to the CSC. Metrology should support the decision to move data to the cloud. Too often, this decision depends on how the risk of adopting CSP security guarantees will reduce the overall cost of data storage and the requisite services. CSCs should have greater transparency as to whether those benefits can be enjoyed while simultaneously avoiding the inherent risk of exfiltration, extrusion, and/or destruction [20].

Data Security Measures

The Cloud Security Association Consortium (CSAC) advises using established best practices to secure cloud-stored data, but challenges persist, especially regarding data security assurance [21,22]. Cloud data insecurity can be attributed to the lack of (i) strong controls, (ii) shared resources, (iii) multi-tenancy, and (iv) visibility. Tailored encryption and access control measures are essential, yet they can inadvertently expose sensitive data.

Data is typically categorized into basic, confidential, and highly confidential levels, each requiring corresponding encryption methods. For instance, basic data or the lowest level of confidentiality can contain data such as videos and photos that are not confidential for the CSC organization and therefore only need a lite version of encryption such as AES-128 encryption, while data at higher levels of confidentiality must be protected with stronger encryption while AES-256 and SHA-3 can be used to prevent unauthorized access in those cases.

Protecting the CIA of cloud data presents challenges. Various methods have been proposed to ensure data integrity, such as a distributed cryptographic system designed to ensure the integrity of cloud-stored data [23], and Barsoum's multi-copy provable data possession protocol [24]. Juels' encrypted disguised blocks for retrievability verification [25], and Shacham's inclusion of pseudo-random functions and BLS signatures [26] provide further examples. Other similar protocols are proof of retrievability mechanisms for data stored in the cloud [27–29]. While these approaches enhance security, some limitations remain, such as the inability to support block insertion operations and a lack of update thresholds.

CSP trustworthiness is another key concern [2,3]. Transparency regarding backup policies, storage locations, encryption, and access control mechanisms is crucial. Shynu proposed a transparent data deduplication mechanism to enhance provider transparency [30] enabling consumers to track where their data is being stored and manipulated. Moreover, Wang has explored error correction capabilities enabling better more resilient data recovery [31].

Renuga investigated data remnants after consumers accidentally/intentionally deleted their data [32] and Han proposed a transparent data sanitization approach for data confidentiality [33]. The approach is implemented as an internet gateway within the premises of the consumer organization using a JavaScript injecting technique that sanitizes sensitive data. Similarly, John proposed an optimal data sanitization algorithm to address data migration issues [34].

2.3. Information Flow Tracking (IFT)

Secure access control models encompass Mandatory Access Control (MAC) and Discretionary Access Control (DAC) systems [35]. While DAC systems, such as Access Control Lists (ACLs) and Role-Based Access Control (RBAC), grant data owners' discretion over access permissions, MAC systems, particularly within the IFT model, are centrally administered to define comprehensive security policies governing data propagation.

In the context of IFT, the administrator defines system-wide security policies, including the labeling of data to control its propagation. Unlike DAC models, which primarily focus on access control within applications, IFT ensures dynamic control over data flow based on predefined security labels [36]. One more option that adds value is IFT's ability to make consumers' data be limitedly labeled [37].

There are two types of information flow tracking systems: (i) Centralized Information Flow Tracking (CIFT) and (ii) Dynamic Information Flow Tracking (DIFT) [38]. For CIFT, the labeling of data is controlled by a centralized empowered body. In contrast, the Dynamic IFT system introduces new labels dynamically into the system at runtime, with an assumed mutual lack of trust and a changeable/dynamic authority [39]. Creating labels and controlling data publishing within one application or overall applications are allowed by DIFT among owners of the data. To prevent data leaking inside one application or through any set of applications, IFT may be used [40] which provides significant advantages from IFT augmentation.

At the level of a programming language-empowered protection domain, IFT can be enforced. Guo's tools to add DIFT-related observations into the source code and perform a static analysis of the data is one example [26]. This form of execution provides resiliency, mobility, and precise control at the byte level. At the protection-domain level, tools that

support the IFT application and enable CSCs to denote rules that can be interpreted and enforced at the process level include Asbestos [27], HiStar [28], and Flume [29].

Asbestos [27] holds significant relevance in the context of cloud environments, especially for information tracking and data confidentiality, despite its primary focus on process and page management. Integration of Asbestos with cloud monitoring systems offers valuable insights into process-level activities, enhancing visibility and control over data flows. This integration empowers administrators to effectively identify and respond to potential security threats or anomalies, contributing to improved overall security and information tracking within cloud environments.

Flume [29], focuses on the domain protection level, where DIFT works on setting the rules that either permit or deny cross-domain information flow and the rules that modify security tags if such flow occurs. At the process level, the performance of using DIFT is important for heavy inter-process communication situations if they occur in an application. Thus, Flume typically causes a performance overhead of over 30% on real applications [41]. The use of IFT rules can provide security against insecure behaviors that may not be prevented by, for example, the DIFT system as well as transparency into the class of secure programs that are implemented by the DIFT system with unmodified semantics [42].

In another example, duPro [41] has proposed an improved user-space DIFT protection domain, featuring an effective framing that empowers implementations to manage the flow of information between components. The result confers a proof of concept for improved security as well as the ability to throttle overhead. Protection domains can be instantiated into the same process that isolates software-based faults for protection domain isolation.

2.4. Studies Using IFT for Cloud Security

Several different IFT approaches have been developed to enforce cloud security in the literature. However, these approaches have not comprehensively addressed the CSC's perspective and the importance of the IFT. In fact, an important fine-grained IFT system has been proposed by Yuan et al., that can expose malware in the cloud [43].

Figure 1 juxtaposes the components described herein that implement the fine-grained IFT system infrastructure. The system tracks the taint using mechanisms that depend on security prerequisites defined by a configuration file relying on (i) script, (ii) source triggering mechanism, and (iii) a method for detecting customizable security violations [43–46]. Many IFT-based mechanisms have been reviewed by Bacon et al., who identified the various challenges necessary to apply IFT in a cloud environment [40]. Their analysis showed that using IFT to provide better cloud security includes the following benefits which are also presented in Figure 1: (i) policy specification, (ii) translation and enforcement, (iii) auditing, and (iv) digital forensic capability. In addition, IFT has the property that can identify vulnerabilities without having to attack (i.e., red team) target systems in the process. Dynamic taint propagation has been used successfully for the same purpose [46].

As well, IFT for IoT applications privacy assessments has been studied by Marcel et al. [47]. Their work enables auditors to model IoT data flow and verify privacy constraints automatically. Fu has conducted research on the scalability of IFT for distributed systems. Fu presented a Dynamic IFT system called "DisTaint" to protect privacy and detect data leaks. Likewise, some studies have proposed Pileus, a system normally used to prevent unauthorized resource access [47]. Notably, their approach applies a Dynamic IFT model to prevent others from conducting vulnerability scanning as this constitutes the first step to unauthorized access to other users' data. Papagiannis and Pietzuch have introduced CloudFilter, as illustrated in Figure 2. This figure overviews the basic steps that demonstrate the process of file upload between a consumer enterprise network and a cloud service provider. In step 1, the user submits the file via a web form, with the browser plugin including user identification and file metadata in the outgoing HTTP request. In step 2, the client proxy intercepts and inspects the request, and matches it against predefined rules in its policy store. Step 3 involves querying the user about the file's confidentiality which results in a

label attached to the file before forwarding the request to the cloud service. Step 4 sees the service proxy examining the request and its labels against local policy rules that potentially deny sensitive files from certain sources. If accepted, the request is sent to the storage service. Finally, step 5 retrieves the uploaded file, with the service proxy using attached labels to guide its response [48]. This mechanism is therefore designed to enable enterprises to retain control over their sensitive data while granting employees access to and usage of enterprise cloud services [49].

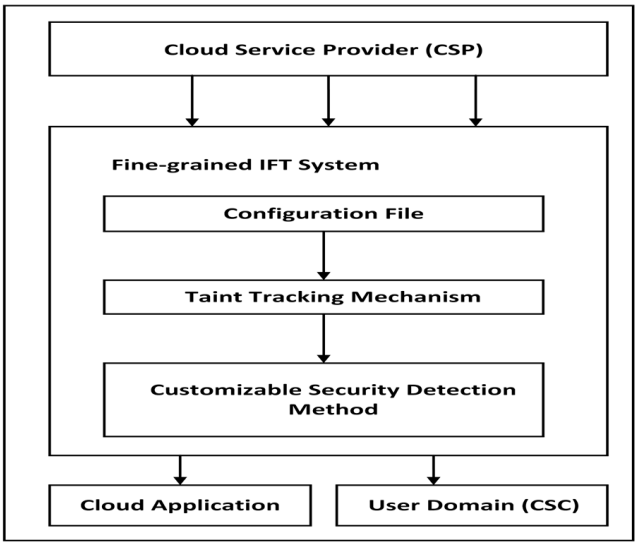


Figure 1. CloudTaint—Fine-grained IFT System for cloud applications, featuring customizable taint tracking mechanisms.

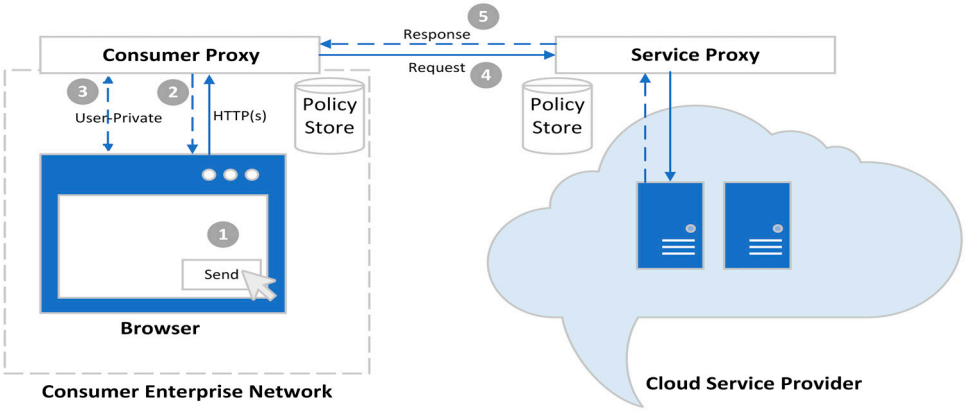


Figure 2. CloudFilter—Dynamic IFT model defending against vulnerability.

Wang et al. proposed a distributed system for IFT that can test many applications, especially for those that would otherwise be resource-intensive and expensive to run routinely [48,50,51]. A model orthogonal to our work (i.e., CloudMonitor), CloudFence was proposed by Vasilis [52] and is shown in Figure 3 as a generic overview in contrast to CloudMonitor. The main interactions within this model are as follows: Users register with the cloud provider (1), and access services from various providers using the same credentials (2). Sensitive data is tagged and tracked transparently across the cloud infrastructure (3). Users then can audit their data through a web interface provided by the cloud provider (4), ensuring control and accountability. Various studies have proposed Data Flow Tracking (DFT) as a service model that supports both consumers and providers in auditing security parameters pertaining to data existing within the cloud. However, this particular DFT model concept is different from our CloudMonitor framework because we do not

prevent the CSC from having the right to audit the data within our framework (i.e., the audit functions are hard-coded and immutable).

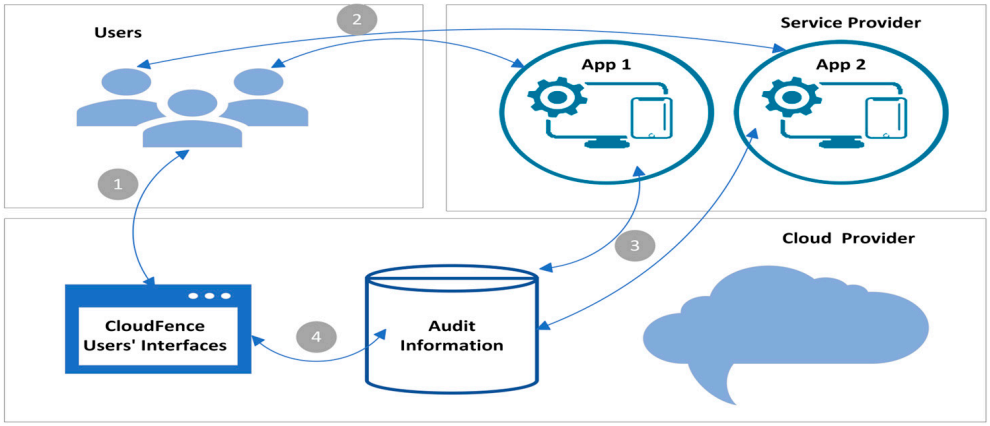


Figure 3. CloudFence—an orthogonal model to our CloudMonitor framework, illustrating the dynamics of advanced cloud security solutions.

Also, the CSP is responsible for maintaining the CloudFence, which is accessible by the consumer as with any other service. Our focus with the CloudMonitor framework is on prioritizing the security needs that customers expect from service providers. In this way, CloudMonitor can overcome the distrust that may exist, and that then requires the consumer to be responsible and thus cognizant to protect their own data in the cloud (i.e., from an auditing standpoint, non-preventive). Our proposed framework considers the actions of users on-site. This allows consumers to validate the reliability of data transmitted from the CSP to their local site. CloudMonitor then ensures that consumers can protect their data both in the cloud and at their premises. There are other instances where IFT systems are implemented similarly using cutting-edge technologies for security purposes but in different contexts such as SDN [53], and IoT networks [38,39].

Secure and resilient access protocols are essential for the data flow that provides access to cloud data. Numerous studies have devised techniques to achieve secure access to the cloud, but only a limited number of studies have utilized IFT. FlowK (i.e., Information Flow Control Kernel Module) offers a persistent security mechanism for CSPs utilizing IFT [54]. At the application level, this technique compels detailed security policies. The system is tested via a framework prepared to deploy IFT-aware web applications in the cloud. John investigated an IFT system that simplifies data protection for tenants and PaaS providers [33]. The results suggest that DIFT is suitable for protecting data integrity and privacy in PaaS platforms.

Bowers' study concentrated on protecting data in the cloud according to its geographical placement [20]. One primary concern for CSCs is the jurisdiction governing the data stored in the cloud. To address this issue, Awani et al. investigated the use of IFT to oversee and regulate the movement of data among various components or applications within the cloud [54]. The primary goal of this study was to assign labels and/or tags to the data belonging to various users to ensure traffic isolation [35].

Leuprecht proposed safeguarding shared data between applications in the cloud using IFT, emphasizing its role in enforcing data flow policies without requiring unique sharing mechanisms [55]. Sun's research also proposes a cloud service architecture utilizing a unique type of IFT designated distributed IFT (dIFT) that isolates user activities and prevents unauthorized access; dIFT thereby protects consumers' data confidentiality, integrity, and especially, CSC intellectual property [56].

Shyamasundar et al. demonstrated the forensic readiness of IFT-based hybrid cloud services. In their demonstrated case, the IFT provides the minimum necessary forensic information from hybrid services [22]. Additionally, their Secure-ComFlow model enhances

cloud security by leveraging IFT acting as a CSC agent for migrating data from local infrastructures to the cloud, allowing users to define their IFT policies for data protection.

3. Materials, Methods, and Tools Used by the CloudMonitor Framework

Herein, the IFT, its components, and its architecture are broadly discussed. Successively, different tools used for the implementation of IFT are thoroughly studied and summarized. Likewise, the requirements for employing the tools have been covered, including the data tagging mechanisms, CPU, and memory requirements. In the end, an experiment that allowed us to comparatively evaluate the Intel-pin and LIBDFT (i.e., the name of a DIFT framework) has been conducted. To study the overhead created by these tools, the IMBench's bandwidth benchmark (i.e., a set of applications for intermittently powered devices (such as wireless sensor networks and IoT devices often rely on intermittent power sources such as ambient light, vibrations, or temperature differences), was used to evaluate the impact caused by both tools over a network. This impact was compared to a native system. The results show the Intel-Pin tool enabled better network bandwidth and throughput compared to the LIBDFT tool. Nevertheless, both tools need improvements to reduce the impact on network bandwidth and throughput.

The rest of Section 3 is structured so that Section 3.1 discusses the different architectures of (DIFT), while Section 3.2 introduces Intel Pin, a dynamic binary instrumentation framework that supports commonly used IFT applications like LIBDFT. The LIBDFT details are provided in Section 3.3. Lastly, Section 3.4 covers the constraints of the present LIBDFT implementation.

3.1. Dynamic Information Flow Tracking (DIFT)

DIFT is used to tag, track, and verify the authenticity of information flows (which may represent nefarious activity such as exfiltration, destruction/disruption, or other malicious activity), such as logging into a computer system from, assumed to be, untrusted input channels. The basic DIFT mechanisms correlate well with tagging and tracking interesting data as they propagate during program execution. The process of "DIFTing" is characterized by three primary flow tracking aspects described here and detailed in the following three subsections (Sections 3.2–3.4).

1. Data sources are usually represented by either a program or a memory location, and they come into play after a function or system call has been made.
2. Data tracking is a process of labeled data tracking during program execution as they are copied/moved and/or altered by program instructions.
3. Data traps are either program or memory locations where the existence of tagged data can be verified for data flow inspection and/or policy enforcement.

DIFT can serve as a valuable tool to support malware prevention, functioning as a detection mechanism for zero-day vulnerabilities, cross-site scripting, and buffer overflow attacks. Moreover, DIFT can be instrumental in identifying and preventing information leakage by implementing tags.

A tag is metadata that indicates the sensitivity of data flow. DIFT tags from untrusted sources as potentially malicious [57]. The status tag provides information that propagates through the system relying on predefined rules, either dataflow or data and control-flow-based. DIFT inspects tagged flows at places known as "data sinks" (or "traps"), to determine if a tagged information flow indicates any malicious activity. Logic related to DIFT can be injected into the original program code in two different ways described here.

3.1.1. Static Information Flow Tracking

During development, Static Information Flow Tracking adds IFT logic to source code and requires compiler modifications (see Figure 4).

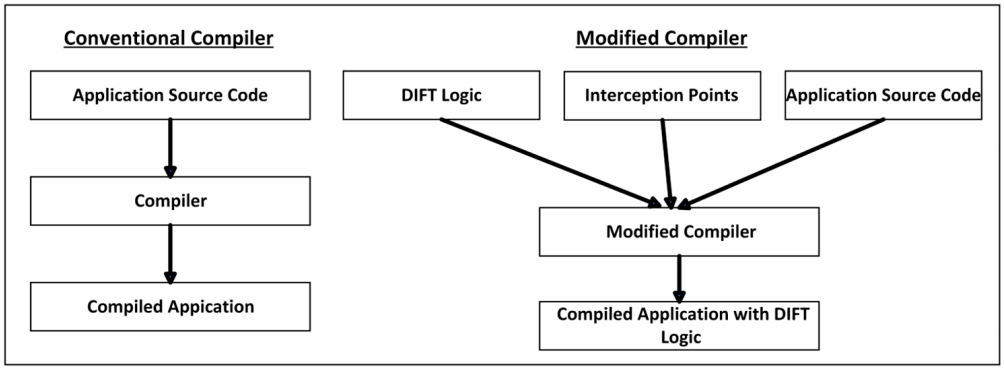


Figure 4. Conventional vs. Static Information Flow Tracking (modified) compiler.

3.1.2. Dynamic Information Flow Tracking

During the runtime of a program, IFT is achieved. Meaning, DIFT logic will be injected into the original program instruction sequence (or flow) with the help of a Dynamic Binary Instrumentation (DBI) framework (e.g., Intel Pin, DynamoRIO) while a program is running. Both Static and Dynamic Information Flow Tracking methods come with their benefits and drawbacks (see Table 3). Throughout this presented analysis, the utilization of DIFT is emphasized.

Table 3. Static Versus Dynamic Information Flow Tracking.

Dynamic Information Flow Tracking	Static Information Flow Tracking
During the application run-time instrumentation code is added	Instrumentation code is added during the compilation of the application
Can be directly applied to any software	Naturally, cannot be applied to any software
Can establish a comprehensive set of guidelines for tracking data flow or a set of rules can be formulated specifically tailored to a particular system/application	Application-specific (when a group of rules specifically functions for a single application)
Performance: Slow, requiring a DBI tool to be attached to the running application to inject DIFT logic	Performance: Faster than DIFT. No DBI tool is required and DIFT logic is already bundled within the application

3.2. Intel-Pin

Intel-Pin is a dynamic binary instrumentation framework specifically catered towards the Instruction Set Architecture (ISA) 32 (32-bit), x86-64 (64-bit), and MIC (Many Integrated Core) ISAs that enables software developers to build personalized tools for dynamic program analysis [4]. These tools, known as Pin tools, can be utilized to analyze user space applications across Linux, Windows, and Mac operating systems. By performing run-time instrumentation of a program's binary file, Intel Pin tools allow in-depth analysis of the program's behavior and performance. As a result, the tool eliminates the need for re-compiling the initial source code and can assist in the instrumentation of programs that involve dynamic code generation (i.e., a relocatable image).

The Intel-Pin API (Application Programming Interface) is well-organized and documented offering an abstraction of intricate instruction-set information. Thus, the API enables contextual details, such as register contents, to be passed as parameters into the injected code from the pin tool. Moreover, Intel Pin has built-in capabilities to save and restore register values that become overwritten by the injected code. This ensures that the application can proceed with its execution as intended, without any disruptions [4].

A pin tool is comprised of three main components: instrumentation, analysis, and callback routines. Instrumentation routines are normally triggered when code that has not undergone recompilation is on the verge of execution, facilitating the analysis routines

insertion. Essentially, instrumentation routines involve examining the binary instructions within a program to decide where and how analysis routines should be inserted. Analysis routines come into play when the code they are associated with is actively running. Callback routines, on the other hand, are triggered when specific conditions or events within the code occur, serving the purpose of security analysis [4].

Therefore, after loading into system memory, Intel Pin performs program instrumentation by taking control of the program. Then, before execution, a JIT (Just-In-Time) compiler recompiles small sections of binary code. Consequently, there are new instructions introduced into the recompiled code, typically sourced from the Pin tool, which conducts the analysis. In addition, Intel Pin has a vast array of optimization techniques to achieve minimal run-time and memory overheads. At the time of this writing, about 30% without running a pin tool is the average overhead of Intel Pin [58]. As anticipated, the quality of the pin tool written by the software developer will greatly affect the overhead.

3.3. The LIBDFT Meta-Tool Capabilities

LIBDFT is a meta-tool that works as a shared library that applies DIFT using Intel Pin's DBI framework [12,59]. LIBDFT offers an API for developers to construct DIFT-enabled pin tools that can function with unmodified binary programs running on standard OSs and hardware. Moreover, the versatility and reusability of the tool make it ideal for research and rapid prototyping purposes.

LIBDFT was specifically developed for use with the Intel Pin DBI framework to facilitate the creation of custom Pin tools. It harnesses the power of Intel Pin's VM alongside a specialized injector component, enabling the attachment of the VM to an existing process or a newly initiated process. For inspecting and modifying a binary executable image, the LIBDFT library relies on the Intel Pin's extensive API. Intel Pin's injector component first injects Intel Pin's runtime and then hands over control to the LIBDFT-enabled Pin tool. This final step is initiated when a Pin tool with LIBDFT enabled attaches to a running process or launches a new one. The LIBDFT library comprises three main components (See Figure 5):

1. Tag map,
2. Tracker,
3. I/O Interface.

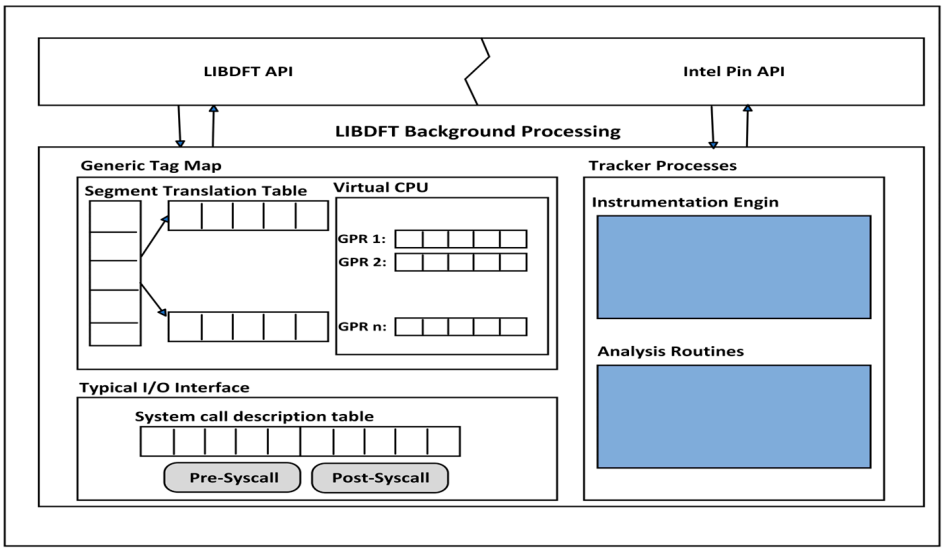


Figure 5. LIBDFT architecture when configured with bit and byte-sized tags.

The use of the Tag map is for storing tags that contain a process-wide data structure also called shadow memory for maintaining the data stored tags in the main memory and the CPU registers. In the tag map, the tags' stored structure is mainly controlled by the

tags' granularity and size. The overall organization and structure of which are shown in Figure 5 [12,59].

Regarding the tagging granularity, LIBDFT can be configured for tagging data as needed (i.e., as small as a single bit or as large as memory chunks). A single-bit tagging result can make fine-grained and accurate DIFT, while larger contiguous memory chunks can cause more error-prone DIFT results. Unfortunately, storing excessive granular tags results in significant memory usage penalties. For example, at the bit level, 8 tags are required for a single byte and 32 tags for a 32-bit CPU register. Thus, LIBDFT can use byte-level tagging to manage memory costs for bit-level tagging. Using byte-level tagging is more sensible for modern CPU architectures, as bytes are typically the smallest addressable memory unit in such modern CPUs. Moreover, this approach makes sufficiently fine-grained tagging for all cases and provides a better balance between usability and performance.

Bit-size and byte-sized tags are offered by the LIBDFT. The aim of bit-sized tagging is to create memory-conserving LIBDFT-enabled Pin tools. This scheme is more desirable since it can enable software developers to 8 varied values to each tagged byte to depict different tag classes. Thus, the byte-sized tags enable more advanced LIBDFT-enabled Pin tools. Next, in-memory data structures are described below in Sections 1–3. The Tag map implementation can be further broken down as described here into the following constructs (Sections 3.3.1–3.3.4).

3.3.1. Virtual CPU (VCPU)

Within LIBDFT, the VCPU structure is employed to store tags for each of the 8 General Purpose Registers (GPRs) in a 32-bit CPU. Various VCPU structures are held by the Tag map. That means, one VCPU structure for each created thread while executing a program. LIBDFT is specially implemented to capture the creation of thread events including termination and to manage many VCPU structures during the specific execution time of a running program. The LIBDFT gives a virtual ID to uniquely identify VCPU strictures related to a certain thread. If LIBDFT is set up to use bit-sized tags, it allocates one byte of memory to store four one-bit tags required for each 32-bit General Purpose Register (GPR). Consequently, the memory needed for each thread amounts to 8 bytes. Conversely, if LIBDFT is configured to employ byte-sized tags, 4 bytes for each 32-bit GPR are required by LIBDFT. Thus, for each thread, 32 bytes of LIBDFT overhead are required.

3.3.2. Memory Bitmap (Mem-Bitmap)

In LIBDFT, Mem-bitmap is used to tag existing data in the main memory of the computer. When LIBDFT is configured to use bit-sized tags, it uses specifically this data structure. Mem-bitmap holds one bit per CPU addressable memory byte because it is a flat fix-sized memory.

3.3.3. Segment Translation Table (STAB)

Similar to the memory-bitmap structure, in LIBDFT, the STAB structure is employed to label data that already exists in the main memory. In contrast to the memory bitmap, when LIBDFT is configured to utilize byte-sized tags, it utilizes STAB. The tags are stored within dynamically allocated tag map sectors by STAB. Each time, a chunk of memory is requested by the program via utilizing a system call such as mmap, malloc, or shmat. LIBDFT intercepts the system call and subsequently assigns memory chunks of equivalent size in a contiguous manner. In the initialization process, LIBDFT assigns the STAB to correlate main memory addresses with their respective bytes, as memory is allocated to a process in blocks referred to as pages. Thus, a page will be pointed to each STAB entry. Crucially, implementation of the LIBDFT guarantees that matched sectors with contiguous memory pages are also contiguous. The technique can help to ease the known problem of memory accesses crossing boundaries (i.e., page fault). Regarding the drawbacks, memory suffers significantly high overhead when using byte-sized tags with STAB.

3.3.4. Implementation of LIBDFT Using Tracker and the I/O Interface

The main component of the LIBDFT library is the Tracker which is responsible for instrumenting a program to insert DIFT logic. The Tracker consists of two components:

- The Instrumentation Engine (IE).
- The Analysis Routines (AR).

The Instrumentation Engine (IE) is responsible for examining a program's layout and sequencing to identify which analysis routines should be inserted into the program. LIBDFT depends on Intel Pin's IE to check each instruction for type, category, and length. First, LIBDFT identifies the type of instruction (e.g., move, arithmetic, or logic instruction), then analyzes operands of the instruction (e.g., memory address, register-based or immediate) to decide their category, and finally the instruction's length (e.g., word or double word). After gathering this data, LIBDFT utilizes Intel Pin to insert a suitable analysis routine ahead of each program instruction. This ensures that the instrumentation code is executed for each execution of the instruction sequences.

The Analysis Routines (AR) comprise the actual code responsible for implementing the DIFT logic for each instrumented instruction. The IE injects the analysis routines before the program creation. Analysis routines are executed more frequently when compared to the instrumentation code. To put it differently, the AR code is inserted to target a particular type of instruction, meaning it will run every time that specific type of instruction is executed. The types of instructions enhanced by the AR can be categorized into the classes outlined in Table 4.

Table 4. Classes of Instructions Analyzed by the AR.

Instruction Class	Description	Examples
ALU	Calculate arithmetic result	ADD, SUB, DIV, IMUL
XFER	Transfer data from register to register, register to a memory loc. and vice versa.	MOV
CLR	Clear/zeroing registers	AND, XOR

The Input and Output interface is tasked with processing the transfer of data between the kernel and the various system and application processes via a variety of system calls. The I/O interface contains a table of system call descriptions "syscall-desc", that has all 344 calls' information of the Linux system. Moreover, the Kernel stores user-defined call-backs and argument descriptors per system call, whether reading or writing data on memory. During program execution, when a system call is triggered, user-defined call-backs are activated both upon entering the system call (i.e., pre-syscall call-backs) and upon exiting it (i.e., post-syscall call-backs).

3.4. Intel-Pin and LIBDFT Tools Evaluation

Here, in this evaluation, tools are compared to understand their performance impact. Several micro-benchmarks are performed by using the IMBench Linux 3.0 performance analysis suite [60]. The testbed employed here consists of two VirtualBox-based virtual machines (VMs), each hosting Ubuntu Linux. The VMs are given one virtual CPU and one virtual RAM. The tools include Pintool and LIBDFT. CloudMonitor achieves IFT by employing such tools. Hence, the importance of considering individual tool performance is obliged. For that reason, the native operating system is compared against the performance overhead created by both the Pintool and LIBDFT tools.

Thus, the IMBench's bandwidth benchmark was used to study the impact caused by both tools on the network data. IMBench monitors the Transmission Control Protocol (TCP) bandwidth while data is moving between the server and the client software from the user side. Different data block sizes were repeatedly transferred so the measurements were repetitive for each block size to ensure consistency, precision, and accuracy. All the

tools were employed using their default settings. Table 5 shows the cost of using both tools versus without (native).

Table 5. Bandwidth Comparison Using Imbench's (In Mb/S).

Data Block Size in Bytes	Native		Pintool		Libdft	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
120	0.20	0.010	10.20	0.20	8.10	0.12
256	0.43	0.021	15.10	0.50	10.43	0.31
512	1.50	0.32	34.15	2.11	27.40	0.50
1024	2.21	1.50	100.23	15.23	75.30	10.30

The results in Table 5 are presented using the mean value and standard deviation in MB/s for the different block sizes. Table 5 shows the differences in the bandwidth across the three cases. The leftmost column gives the size of the data transferred between the server and client for repeated rounds. The result is shown as the impact on throughput over the bandwidth. Bandwidth and throughput are two network performance-related terms. Bandwidth relates to network capacity while throughput describes the amount of data transmitted.

4. CloudMonitor as Our Conceptual Framework

The CloudMonitor framework is a pragmatic approach that can monitor and control data flow between the consumer and the CSP. Moreover, CloudMonitor enables consumers to review their data housed in the cloud. This is achieved by connecting data propagation policies through the utilization of security labels embedded within the data intended for transfer. Thus, relying on these consumers' labels can control how their data propagates when initially transferred or uploaded to the cloud service separate from the CSP. Additionally, the user's browser and the client API of the CSP service are modified in a way that detects every upload that is being executed to monitor employee actions in relation to the data they have placed in the cloud.

The next part of the CloudMonitor is a service provided by CSP to the consumer. For this framework part, the CSP incorporates a simple API, whereby DFT is integrated into their offered services. With the DFT API, a CSP can label any sensitive data that requires additional protection. The primary objective of this second capability is to guarantee that when a user uploads data to the cloud, all operations are meticulously recorded, actions are linked to the authorized user, and access to this data is restricted to specific network groups. This can give more confidence to consumer organizations in adopting a CSP for a portion of their data transfer and storage requirements, at the very least.

Likewise, the CloudMonitor framework is prepared to support consumer organizations with many employees who would concurrently propagate significant amounts of data that do not carry the same tags. As a result, the data tracking level is executed at a level of fine-grain to cover any sensitive data requirements through a complete set of information tracking constructs, and the "Time-Controllable Keyword Search Scheme" for mobile e-health clouds has more specific focuses.

The CloudMonitor conceptual framework, depicted in Figure 6, outlines the process of data contamination and subsequent tracking within cloud environments through three main steps:

1. **Tainting Data at the Source:** Initially, sensitive information is labeled or "tainted" at the points of entry where it is transferred to the cloud. This step involves monitoring the flow of data both into the cloud (to prevent leakage) and within the consumer network (to detect potential contamination, such as malware). Key points of potential data leakage, such as Cloud-client APIs, user browsers, and local consumer premises, are identified as "taint sinks" where data inspection occurs to assess sensitivity and take appropriate actions.

2. **Enforcing Propagation Policy:** The propagation policy governs how tainted data moves from the sources to the sinks. It determines the granularity and precision of taint tracking, ensuring that data is tagged at the most detailed level to achieve appropriate policy-driven tracking accuracy and reliability.
3. **Sanitizing Data at the Sink:** Finally, data is sanitized or reviewed for sensitivity at the sink points before further processing or storage. This step ensures that only authorized data is retained and that appropriate actions are taken to mitigate any risks identified during the tracking process.

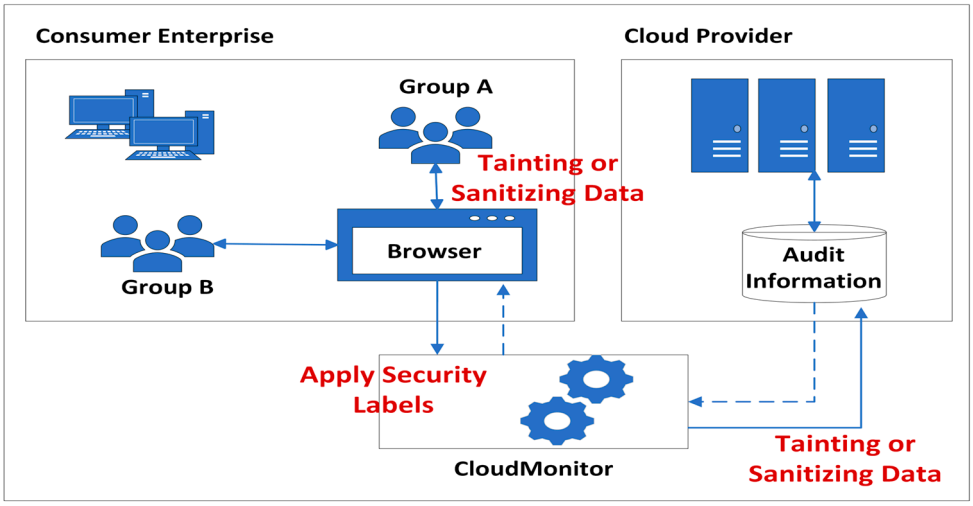


Figure 6. CloudMonitor Framework is implemented as an experimental testbed.

By following these steps, CloudMonitor provides a comprehensive framework for monitoring and controlling data flow within cloud environments, enhancing security and data integrity for both CSCs and CSPs.

4.1. Design Goals of the CloudMonitor Framework

Usability, deployability, and flexibility (malleable and extensible) for IFT are the main goals in designing the CloudMonitor. Facing a variety of security needs that are required from both the consumer and provider sides is a must for the CloudMonitor framework to enable tracking data within a complex cloud ecosystem. High-level security protections are essential. While, at the same time, such protections must not create high-performance overheads that render the CloudMonitor slow, inefficient, and unusable. Consequently, the CloudMonitor framework must be consistent with current applications and operating systems. Additionally, the framework needs a balance in the propagating and tainting process, ensuring efficiency while maintaining high assurance reliability.

Another framework prerequisite is that it can be able to enforce the isolation of data. In this manner, the framework prevents data exchange among (unauthorized) applications. Also, the framework needs to taint any data coming from the cloud, at the local network level of the consumer. This facility can thus identify memory locations or arguments of certain data that are tainted. For instance, the cloud URL strings that can be visited by users might be categorized as a taint source. Additionally, the data in the cloud received by the browser in response to up/down transfer requests is also tainted. That means HTTP pages and the data fetched from the cloud are considered tainted according to the requirements of the defined policies.

4.2. Tools Selection Overview and Rationale

In this section, a summary of the works that would support DIFT being adopted within the cloud is provided. IFT systems come in both hardware and software-based varieties [61,62]. Implementations of Hardware-based IFT are beyond our scope of work.

Operating systems enforce the IFT systems of software-based include IFT. At the process level, in an operating system (OS) based IFT, data tracking is conducted, labeling the processes and continuous data. Thus, each time accessing persistent data, and if inter-process communications took place, then propagation of tainted data occurs. Asbestos [47] and Flume [50] exemplify an operating system-based IFT enforcement process. While Flume is a program to run on top of the Linux OS, Asbestos is an OS that can enforce IFT. DStar [63] interprets labels of the security between instances that can able IFT in distributed systems. Also, Aeolus offers tracking of the IFT of communication cross-host [64]. However, to achieve this capability, Asbestos is employed as a distributed OS.

Unfortunately, Flume suffers from security issues inherited from its underlying architecture and design choices, including vulnerabilities related to data integrity, authentication, and access control. These issues arise due to limitations in encryption mechanisms, inadequate authentication protocols, and insufficient access control measures. DStar can be used appropriately in the cloud, and given its nature, works across a range of OSs like Flume and Linux. Aeolus can operate as a trusted computing base layered above Asbestos. Furthermore, to run distributed communication, Asbestos is extended by Aeolus. That means, it makes applications run on a trusted base (i.e., Input/Output filtering, exterior connections, and inter-thread), thus enforcing any data-related policies.

In addition to the hardware and OS-based IFT systems, there are also IFT systems that operate at the middleware level, such as DEFcon [40] and SafeWeb [65]. The latest, SafeWeb, acts to ease policy breaches in multilevel applications. Moreover, SafeWeb confirms data privacy and safety across all various web app levels if tracking data flow by using IFT.

Returning to the topic of Storage as a Service, which CloudMonitor supports, let us explore how these various IFT schemes that have been discussed support CloudMonitor utility. Specifically, our research here mainly uses the IaaS cloud service model. Consequently, we track only data flows that reside on cloud virtual machines (VMs). Chief significance in this context is a fine-grained distributed Information Flow Tracking (IFT) system capable of monitoring data flow across VMs. Such an IFT application can be easily operational within an organization utilizing IaaS for its cloud infrastructure. Accordingly, this is where the initial workings of the CloudMonitor framework can and was put into practice.

Fitting with the IFT implementations reviewed, the isolation unit uses the tracking granularity at the level of process, thread, and/or object [66]. The next part of the CloudMonitor framework requires the participation of cloud service providers with the IFT strategy. The consumer-extended VMs are exposing labels [67]. Unlikely, the provider directly modifies the security policies required by the consumer to be adopted into the IaaS service.

Nevertheless, the service provider indirectly impacts the data flow exchange among various VMs, whether they belong to the same consumer or other consumers, by utilizing IFT at the network level. CloudMonitor leverages some widely used mechanisms from DIFT systems. As such, CloudMonitor adopts DIFT characteristics from both Flume and Asbestos.

4.3. The CloudMonitor Use-Case

This section describes a use-case for CloudMonitor to present its effectiveness. "OwnCloud" is a collection of client-server software components designed for establishing and using the services of file storage. OwnCloud has a similar functionality compared with the widely used Dropbox or Google drive. We used OwnCloud as the cloud server where consumers can upload and/or download files as desired. Normally, the OwnCloud server can be accessed via a SOAP and RESTful API [68]. In the CloudMonitor implementation, we have only demonstrated this access via SOAP API. Three users have been created on the OwnCloud server to validate the implementation of the CloudMonitor framework. Each user is expected to upload a file via a modified web browser. The browser is modified with a plugin that attaches outgoing HTTP requests with a set of identification information along

with the user and file metadata. The file metadata includes where the file is locally stored. Subsequently, by intercepting the user's request, the API stub inspects the request content and retrieves the user information. It then matches the request against the consumer's rules residing in the policy store.

If a service request fails to match the rules, a message is sent to the user for acknowledgment. In the other case, where the user request matches with the rules in the policy store, then the API stub executes the request. Based on the segment of the network in the local infrastructure of the consumer from which the user is sending the request, (i) the system informs the user about the confidentiality of the file being uploaded. As a result, (ii) the file is either labeled confidential or not confidential. The system API stub then (iii) continues with the file upload request to the OwnCloud server. Here, (iv) the request along with its labels are matched against the service provider policy and a decision is taken on whether to store the data or reject the request. Once the data storage is accepted (v) the data flow tracking (DFT) is performed within the boundaries of a well-defined DFT domain. Whenever the data crosses a specified domain, (vi) the CloudMonitor logs the action in its audit database. CloudMonitor has a user interface that leverages Cloudopsy (i.e., a web-based data auditing dashboard) so (vii) the user can have a comprehensive view of all audit information.

4.4. Enforcing the Rules of Data Isolation

The isolation of consumer data instances in the cloud is demonstrated using the IFT constraints discussed above. Here, different consumers that use the cloud servers are allowed to allocate and assign tags that allow them to share data more safely. For instance, in healthcare, medical records saved in the cloud are accessed by physicians working in several registered hospitals [68]. With this implementation, physicians may have access to their current patients' medical records, and medical researchers can have access to anonymized data sets. Other information may be restricted to the owners of the medical records datasets. IFT enables not only isolation but also flexible data sharing. Coming back to the CSC organization's (CO) data security, the information flow rules with respect to the organization-initiated monitoring are given here. When the data leaves the User1 in LAN1, the data is tagged as follows:

$$\text{Secrecy (User1)} = \text{((User1, LAN1), data)} \tag{1}$$

$$\text{Integrity (User1)} = \text{(consumer\_generated, no\_consent)} \tag{2}$$

Here, the tags show that the data is sent by User1 in LAN1. The data is generated by the CO, and it does not allow its use by others without consent. When the system administrator of the CO logs into the cloud service to check the data, the following application process is created with the tags:

$$\text{Secrecy (LAN1)} = \text{(LAN1, data)} \tag{3}$$

$$\text{Integrity (LAN1)} = \text{(consumer\_generated)} \tag{4}$$

On monitoring the data sent from LAN1, the IFT-aware OS first matches the tags of the data with those of the application process according to Rule 4.1.

$$User1 \rightarrow LAN,\ if\ f\ Secercy(User1)\ Secrecy(LAN1)\ integrity(LAN1)$$

$$integrity(User1) \tag{5}$$

Since the condition is met, an administrator has access to the data submitted by User1. Let us consider the case where a third party (e.g., researcher) needs to access to the data submitted by User2 with the following tags:

$$\text{Secrecy (User2)} = \text{((User2, LAN1), data)} \tag{6}$$

$$\text{Integrity (User2)} = (\text{consumer\_generated}, \text{consent\_research}) \quad (7)$$

The secret tags are removed from the data via an anonymizing process. That is, once the data that is requested for research is consented to, then the anonymizing process comes into play and removes the data tags to achieve the anonymization.

To achieve network-level isolation, the network abstractions of the virtual environment are logically separated in a way that each network area has its own devices, routes, and firewall rules. Each network area can be accessed only by its users via a dedicated bridge using the case-specific firewall rules. That is, each user is restricted to accessing their own network objects within the same network area. Each VM is attached to a separate private network in the environment. Both participants in a pairing are used to connect two private bridges to a host bridge to access the physical network. When a firewall rule is specified, the rule is applied to the private bridge instead of the host bridge, therefore eliminating its potential effects on the other user's VM.

The cloud storage level isolation is achieved by extending the IFT to the OwnCloud server. With IFT, the server assigns tags to stored objects and enforces an access control mechanism over all requests to the stored objects. For each access request, the tag of the requesting user and the objects are evaluated against the security module rules (which codify the appropriate policies) which perform access decisions enabling consistent enforcement of access control for the server and stored data. The server is considered fully trusted because it runs the enforcement mechanism. As previously described in the introductory Section 4, a user requesting access to the server must use the OwnCloud desktop client on the user node. The server will then obtain an OwnCloud desktop client tag. This process prevents unauthorized adversary nodes from accessing the cloud server. This prevents an adversary node from using an OwnCloud desktop client with the tags of any users for which it holds ownership (i.e., to prevent spoofing). The tags are specific to each user; therefore, spoofing will be declined if an unauthorized user tries to access the service by enforcing the policy.

4.5. Results

This section focuses on the methodology, evaluation, and results of our experimental framework. Our CloudMonitor prototype was implemented on a physical machine. Linux operates on the virtualized node's Intel Pentium Dual-Core 2.60 GHz processor and 16 GB of primary memory. VirtualBox has been installed on the nodes to facilitate virtualization. OwnCloud has been installed on the actual machine as a cloud server, and Windows 10 is running on the VM. Both the functionality and performance of the CloudMonitor are evaluated.

CloudMonitor's essential modules are virtual. The purpose is to host emulated execution modules and the integrity detection module. To transition from the virtualized mode of execution to the emulated mode, the native VM must be suspended, a full snapshot of its virtual CPU state must be created, and the emulated processor must be initialized using this snapshot. Using shared memory, there is a collaboration between the hypervisor and the emulator for arranging their activities and exchanging state information. The virtual module is essential for implementing IFT, which dynamically transfers the VM to the emulator without disrupting any cloud application services.

During VM migration, the execution of the user domain reaches the Extended Instruction Pointer (EIP) register, a component of x86 architectures (32-bit), which holds the memory address of the next instruction to be executed. This register is pivotal for tracking the migration process as it signals the point at which the CPU snapshot of the VM is written to the emulator's log file.

If privacy-violating conduct is found, the CloudMonitor rejects activities immediately. For anomalous behaviors that surpass the threshold for suspension, CloudMonitor triggers an alarm and logs the occurrence in the system log for auditing purposes.

Using LMbench [69], we analyze CloudMonitor's performance by selecting metrics of the CPU, processes, file system, and virtual memory system. We first analyze CloudMonitor's performance without tracking information flow. The comparison includes two setups: one with CloudMonitor installed but not tracking information flow (Protected VM), and another with CloudMonitor installed and active only during VM migration (Translation VM). Results show CloudMonitor's overhead is under 10% without information flow tracking and less than 20% during VM migration compared to native Xen (Figure 7).

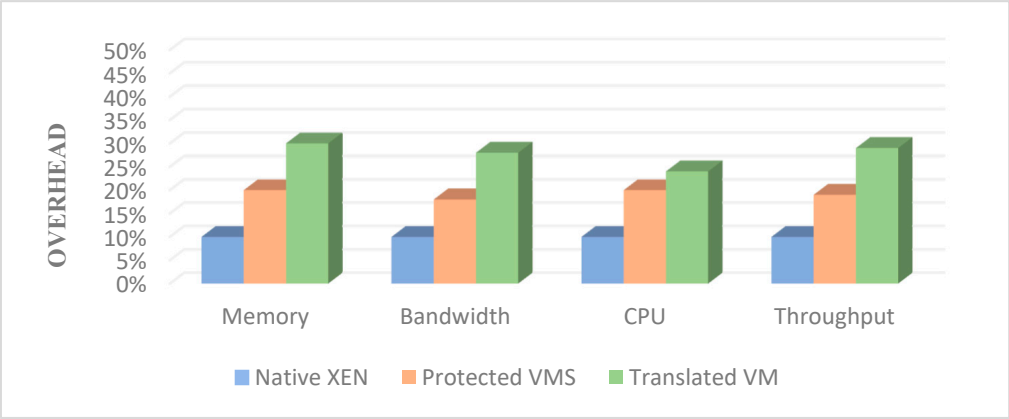


Figure 7. Performance of Native Xen, Protected VMs, and Translated VMs.

The 10 percent performance cost without Information Flow Tracking (IFT) encompasses the cost associated with loading the configuration file based on the security requirements of cloud applications. It also includes the cost introduced by triggering mechanisms for trapping taint sources access and the cost from communication and interaction between the functional modules of CloudMonitor. The 20% performance degradation observed during the transfer of VMs from virtual to emulated execution is primarily attributed to the processes of saving and loading the VM CPU context. Then, we test the performance of the CloudMonitor's IFT when the VM is running as an emulator. TEMU represents the traditional taint tracking system implemented on QEMU [70], and CloudMonitor represents the information flow tracking engine built for our experimental comparisons. The overhead is reduced between 20% and 50% compared to TEMU (Figure 8).

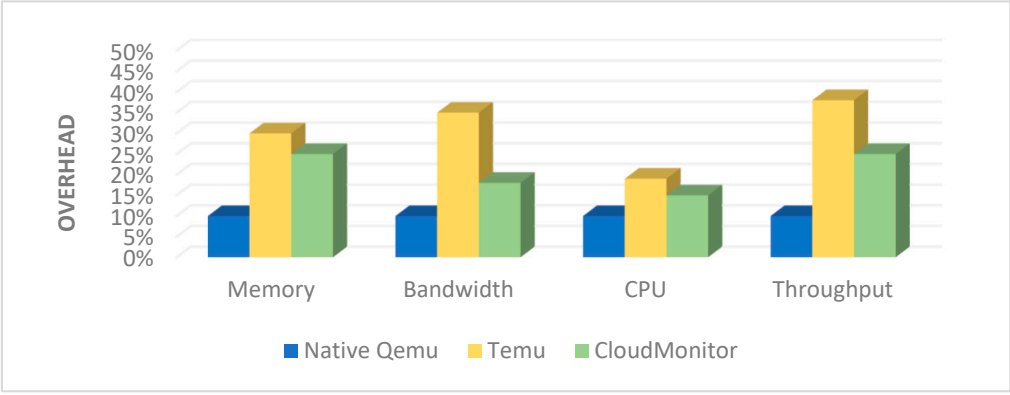


Figure 8. Performance of QEMU, TEMU, and CloudMonitor.

5. Data Leak Prevention Using CloudMonitor

Managing the confidential data flow stored in the cloud means ensuring that only authorized parties can access properly identified and controlled data and documents, which is one of the most important security worries in handling security for an enterprise. A recent significant leaking history of sensitive information has shown that a lot of institutions,

encompassing those within government, education, and business, are extremely lacking in this context [71]. As consumer reliance on cloud computing infrastructures increases, it becomes increasingly tougher to track sensitive information transmission and maintain privacy policies. Given the extensive range and diversity of accessible transferring information channels to users in a normal environment of IT, monitoring the flow of data across all cloud storage instances may appear onerous, if not impossible.

Unauthorized disclosure of private or classified information can inflict significant harm on both organizations and society [72]. Despite substantial investments in cutting-edge security technologies, preventing leaks of sensitive information remains a formidable challenge. Human error, including incidents of data extrusion and social engineering, plays a significant role in compromising security [73]. Users must consistently adhere to dissemination limitations, yet carelessness and impatience often lead to inadvertent compromises in security.

Technology holds promise in identifying and mitigating these negligent behaviors. With appropriate controls and monitoring mechanisms in place, we can enhance accountability and mitigate the effects of human error. However, developing effective and feasible technological solutions requires addressing numerous challenges. These include keeping track of the various ways information is transferred among users and accounting for the diverse methods through which cloud data can be modified, converted, and transmitted.

Most popular operating systems (OSs) and user applications provide minimal support in this regard. Simply put, current security systems for mediating access to data in the cloud are ill-equipped to follow subsequent data changes and the flow of data between tenants. Considering a user who opens a private document in their word processor and edits it using cloud-based data. Then, the document is transferred (moved, copied, or duplicated), within an environment without data leak prevention (DLP) [74] for which there are no dissemination limits, in a rare moment of carelessness. This simple procedure creates another confidential paragraph copy. However, this copy lacks any original document relationship, its confidential status, and constraints on its publishing undoubtedly putting the confidential material at risk of leaking.

In the absence of a thorough software stack overhaul from the ground up, we believe that preventing occurrences of this sort demands an overall and clear platform for user data flow tracking, information exchange tracking, channels, and security standards implementation. Our premise is (1) a comprehensive platform for tracking information flow that is compatible with unmodified programs and OSs is feasible, and (2) specialized middleware offers the most efficient and architectural base for a platform of this nature.

To test this concept, we provide an IFT framework with a unique architecture of security and a group of accompanying methods for tracking fine-grained information flows in cloud storage services. Our overarching objective is to provide a strong platform of information management that enables enterprises to set and implement end-to-end regulations for the propagation and utilization of sensitive information saved to the cloud.

5.1. The Data Tracking Policy Framework

The founding concept, which informs an information recipient and leverages a basic and scalable unit of data granularity for access control is the essence of our policy framework. A policy construct (i.e., principal) might represent a single user inside a CSC organization or a group of users with the same or equal access permissions (e.g., employees in the research department).

The CloudMonitor's method for policy design and reinforcement is dependent on the Dynamic label standard, an effective paradigm of access control that allows more than a principal to secure the sensitive information and be shared under specified conditions.

In the Dynamic label paradigm, each data value has allocated a label that represents a particular set of distribution restrictions. A label conceptually reflects an unorganized collection of privacy policies. Each policy has a designated owner and eligible readers. The owner of the policy related to a data item (d) is a principal whose details have been

identified as having established the value of the data item. This principle also aims to restrict the data's exposure by declaring a policy. The reader set, indicated by the policy, represents the individuals or entities authorized by the owner (o) to monitor and perform calculations on d. An individual principle can appear in numerous readers' sets with their own various policies. In addition, a principal can change (weaken or strengthen) its own policy on a particular data item by modifying the set of readers (r).

As the default behavior, all data items recently created are given an empty label (denoted $L\phi \equiv \{\}$), which does not contain policies and reflects data that is entirely accessible to the public. A data item is considered contaminated if it contains a non-empty label. When dealing with labels that have multiple policies, a principal P may observe data if and only if each policy specifies P as an approved reader. The effective reader set is constituted by the intersection of all reader sets included within a label form.

Consider the data item d labeled to show these definitions.

$$La = \{\{o1 : r1, r2\}, \{o2 : r2, r3\}, and \{o3 : r2, r4\}\}$$

The three policies in this label are owned by $o1, o2$, and $o3$, respectively. The policy of principal $o1$ permits $r1$ and $r2$ to witness the value of d; the policy of principal $o2$ permits $r2$ and $r3$ to observe d; and the policy of principal $o3$ permits $r2$ and $r4$ to observe d. Therefore, in this instance, the efficient reader set comprises the common element $r2$, and as a result, only this principle has access to and control over d.

Consider Alice and Bob as hypothetical employees collaborating on an internal project involving confidential information and serve as concrete examples for discussion. Let us assume that Alice possesses a secret file, denoted as $f1$, which she intends to securely exchange with Bob. To achieve this, Alice can establish a new secrecy policy, denoted as $pA = \{Alice : Alice, Bob\}$, thereby permitting Bob to access, store, and manipulate $f1$ while explicitly prohibiting disclosure to unauthorized parties.

Suppose Bob also possesses a separate file, denoted as $f2$, with the label $pB =$ "Bob: Bob, Charles". At a later point, Bob may seek to integrate the information from $f1$ and $f2$, such as by cross-referencing their contents. The resulting computation yields a new file, named $f3$, labeled with the union of their respective policies: pA, pB.

However, Bob inadvertently attempts to share $f3$ with Charles, overlooking the fact that it contains data derived from Alice's secret file. As per Alice's policy, Charles is not authorized to access her data, highlighting the necessity for the system to prevent such actions and thus prevent data leakage.

Now, assume that a subsequent employee, David, requests authorization to read the relevant files after joining the classified project ($f1$ and $f3$, but not $f2$). To provide David access to file $f1$, Alice adds him to the list of permitted readers in pA. Bob develops a new policy $pB' =$ {Bob: Bob, David} and relabels $f3$, replacing his old policy pB with pB', to make $f3$ accessible. Note that Bob could also make $f3$ accessible to David by adding him to the reader set of pB, but this action would also have the unintended consequence of revealing $f2$ to David. These sorts of conundrums can be avoided using consistency checking, say via a model checker for instance.

5.2. The Information Flow Tracking Mechanism

The CloudMonitor framework observes computational actions made on values of sensitive data at the machine instruction level and publishes labels accordingly. The framework supports operations involving the combination of values of multiple (usually two) unique operands by merging the input label values. Label merging is a core function that generates a fresh data label by combining the policies specified by the input labels. Given a pair of labels $L1 = p1$ and $L2 = p2$, where $p1$ and $p2$ represent arbitrary policy sets, the merge operator (denoted $\bigoplus$) produces a new label corresponding to the union of the input policy sets: $L1 \oplus L2 = p1 \cup p2$. New label A contains the policy sets of $p1$ and the policy sets of $p2$ (A = $p1 \cup p2$). This definition of label merging eliminates the danger of information leakage caused by computations expressed using binary operators. The

resultant label specifies the least restrictive secrecy policy while enforcing all limitations on the input operands utilized in the computation.

Thus, CloudMonitor scrutinizes all clearly defined data movements originating from variable assignments and mathematical operations in the current design. We also monitor indirect flows resulting from pointer dereferencing, in which utilizing the value of sensitive data as a foundation pointer or an offset to access another value in memory.

The CloudMonitor does not handle implicit channels that result from the dependency of control flow, like when a labeled value affects a conditional branch. Such enforcement is extremely difficult to appropriately identify these runtime dependencies without previous static analysis at the source code level.

Next, we explain the concept of instruction-level label tracking and highlight the distinction between explicit and implicit information channels using numerous simple examples. Figure 9 depicts the C-language and assembly-language implementations of the simple function compute sum. This function receives two integer inputs and returns their total, as suggested by its name. Assume that the input variables (a and b) are contaminated with the relevant data labels La and Lb. At the instruction level, CloudMonitor monitors the computation and propagates the following labels when this operation is conducted within the CloudMonitor-managed environment. The first instruction moves the previous stack base pointer value (register ebp) into the stack. In the IFT context, ebp is a control register that typically does not include sensitive user information. Therefore, we do not monitor the labels' propagation into this register and presume that its contents are always non-sensitive. Consequently, this instruction transfers a four-byte value that is not sensitive to the top of the stack, and CloudMonitor removes the sensitivity label linked with the appropriate memory address: $Leme[esp + (0 \ldots 3)] \rightarrow L\phi$. The instruction at location +0x3 moves ebp from the stack into edx, and CloudMonitor assigns the label Lb to edx to track its effects: $Ledx \rightarrow Lb$. Likewise, the following instruction propagates the label La to $eax : Leax \rightarrow La$.

```
int compute_sum(int a, int b){
  return a + b;
}
------------------------------
<compute_sum>:
+0x0 push %ebp
+0x1 mov %esp,%ebp
+0x3 mov 0xc(%ebp),%edx
+0x6 mov 0x8(%ebp),%eax
+0x9 add %edx,%eax
+0xb pop %ebp
+0xc ret
```

Figure 9. The Compute sum function for Instruction level IFT.

The final instruction at position +0x9 computes the total by adding the value in edx to the contents of eax, then CloudMonitor updates the register labels by merging the labels of the two input operands: $Leax \leftarrow Leax \oplus Ledx$. The final two instructions restore control to the caller by restoring the values of ebp and eip from the stack. Since CloudMonitor does not monitor the flow of data through these registers, no further action is required.

Figure 10 demonstrates the implementation of a fundamental table lookup operation and the publication of data labels by dereferencing the pointer. Assume the table contains sensitive values marked with Lt and the input parameter (table index) is contaminated with Li. The assembly code reveals that the lookup process is carried out by two sequential instructions: placing an index from the stack into eax ($off\ set\ 0x3$) and computing a pointer to the corresponding table item and dereferencing it into eax ($off\ set\ 0x6$). In this case, the

instruction at +0x3 contaminates the *eax* register with the argument's label: $Leax \leftarrow Li$. The instruction at +0x6 conducts an indirect memory reference via a tainted pointer, and CloudMonitor handles this by merging the label of the pointer with the memory location(s) label to be accessed: $Leax \leftarrow Leax \oplus Lt$.

```
int table[]={…};
int table_lookup(int index){
  return table[index];
}
--------------------------------
<table_lookup>:
+0x0 push %ebp
+0x1 mov %esp,%ebp
+0x3 mov 0x8(%ebp),%eax
+0x6 mov 0x8049700(,%eax,4),%eax
+0xd pop %ebp
+0xe ret
```

Figure 10. Table lookup function for instruction level IFT.

Correctness is essential regarding the CloudMonitor framework design and implementation. This is especially true concerning pointer access. Table lookups are an incredibly common process that happens in a variety of circumstances involving the manipulation of private user data, such as character set conversion. The inability to monitor indirect data flows that arise from table access can easily result in the undesirable loss of sensitivity status in a wide variety of typical situations.

While further investigation into taint explosion is necessary, our analysis and experience with the CloudMonitor prototype suggest that previous studies' pessimistic findings on pointer tracking effectiveness are unwarranted. As seen in various studies, various simple preventative measures might be implemented to eliminate the proliferation of taint at the kernel level, allowing thorough tracking of all direct and indirect information pathways.

Lastly, the illustration in Figure 11 depicts an implicit information channel that is not monitored by CloudMonitor. Consider that the input argument value v affects the conditional branch instruction at position $+0xa$: if v is not equal to 0, the execution jumps to +0x15; otherwise, the execution continues with the following instruction (+0xc). This function puts an immediate fixed value (0 or 1, relying on the branch) into a temporary memory address and then transfers it to register *eax* in both circumstances. An instant value is case-insensitive; therefore, this function will constantly revert to a value that has been tainted with L, regardless of how the incoming value is tainted. In other words, it leaks a piece of information regarding the entry value v.

Following implicit channels through runtime dynamic analysis can be highly challenging, and most prior systems designed to monitor these channels rely on a type of static analysis performed at the level of source code [75]. In the presence of malicious code, the inability to trace implicit channels is significant, as they facilitate the "laundering" of sensitive data for exfiltration. Currently, CloudMonitor focuses on ensuring the information flows in a safe environment so that implicit flows do not pose a significant challenge. In our early discussions, we emphasized that non-malicious programs seldom leak information implicitly. Our current tracking systems effectively capture all modifications to explicit data in many common applications.

```
int table[]={…};
int is_nonzero(int v){
  if(v==0)
       return 0;
  else
       return 1;
}
---------------------------------
<is_nonzero>:
+0x0 push %ebp
+0x1 mov %esp,%ebp
+0x3 sub $0x4,%esp
+0x6 cmpl $0x0,$0x8(%ebp)
+0xa jne <is_ nonzero+0x15>
+0xc movl $0x0,-0x4(%ebp)
+0x13 jmp<is_nonzero+0x1c>
+0x15 movl $0x1,-0x4(%ebp)
+0x1c mov -0x4(%ebp),%eax
+0x1f leave
+0x20 ret
```

Figure 11. The is-nonzero function for instruction level IFT.

5.3. Implementation

This section focuses on the CloudMonitor's implementation details. The prototype has been designed with a focus on minimal computational requirements to address the limitations of existing solutions, particularly for IFT. To achieve this, CloudMonitor strategically minimizes reliance on Intel Pin and restricts its use of the Intel Pin-based logging tool. This decision aims to mitigate performance costs associated with Intel Pin's injection of instructions into the original code base of the operating program [76]. The prototype is built on top of Linux kernel components such as Netlink and the process event module [77]. These components establish a connection with the kernel space, allowing CloudMonitor to receive notifications of process events (e.g., process creation under Linux operating systems). This design choice preserves the tool's lightweight nature. Furthermore, CloudMonitor's visualization capabilities empower security analysts by providing a clear representation of data leakage via malicious software behavior. Graphs depicting the process trace offer insights into the operation of malicious programs on both the victim's computer and the cloud provider's service.

5.3.1. Consumer Side Implementation Perspective

The implementation of CloudMonitor from the consumer's side aims to provide robust capabilities for tracking information during potential security breaches within cloud environments, particularly focusing on attacks utilizing Secure Shell (SSH) and secure copy (SCP) communication protocols [78,79]. From the consumer's perspective, CloudMonitor facilitates the visualization of the victim system's process trace during the execution of such attacks.

A start and termination script is used to start the logging application on the consumer's end before the attack is launched [80]. This is accomplished by delivering a signal to the victim's computer via the SSH protocol to initiate the Netlink-based logging software. When the attack is complete, a signal via the SSH communication protocol will be sent by the start and termination script to the consumer computers to terminate the process logging 30 s after receiving the signal. These 30 s paddings are added to allow consumer-side processes linked to the attack to complete their jobs if they were still running at the time

the signal was received. This synchronization method (start and stop signals) enabled the capturing of processes created during the attack's time frame.

Next, the Netlink-based logging tool is designed in C/C++ to promptly communicate with the Linux kernel via the Netlink kernel module, allowing for the retrieval of process notifications including their Process ID (PID), Parent Process ID (PPID) [81], and executable information. At the end of the Netlink-based logger execution, a log file is generated containing a record of every process executed on the target system during the attack. This log file serves as the foundation for information tracking.

The visualization component is a Python script that processes the log file generated by the logging instrument. The Python script peruses the log file and initially locates the log entry where receiving the stop signal is achieved. After that, it constructs in-memory dictionary objects for each distinct process using all the available process log data up to the moment when the termination signal was received. Upon completion of its examination of in-memory dictionary objects, the Python script will construct associations between parent and child processes and communicate this information to the Neo4j Database management system [82] using the Neo4j Python driver to generate a process trace graph.

Neo4j is an open-source and graphing database that provides back-end functionality for applications [83]. Neo4j is a native graphing database because it implements the property graph model methodically down to the storage level. Data is saved exactly as it was white-boarded by developers to be highly scalable for this kind of target environment. Thus, Neo4j was selected as the system of database management for the CloudMonitor due to its adaptable data model, high scalability, simple data retrieval using the homegrown graph query language, and real-time visualization capabilities.

5.3.2. Provider Side Implementation

CloudMonitor was designed to provide security analysts with a clear view of the processes generated on the provider's side of the system during an attack. This enables analysts to understand how a malicious program generates processes throughout its operation, facilitating the visualization of the program's process trace during execution. CloudMonitor, from the provider's vantage point, includes both the Intel Pin-based logging tool and the visualization tool.

The Intel Pin-based logging tool is C/C++ software 1.19.9 developed with the Intel Pin tool that enables binding to a harmful program and detecting all operations launched during its execution. When new processes are generated, the logging tool gathers a variety of spawning process information. It logs commands executed by the process together with their parameters, the user to which the process belongs, and the group to which a user belongs. During the investigation of a dangerous program, the logging tool will log this information to a log file, which will then be processed by the visualization tool.

The visualization component, coded in Python 3.13, serves as a post-execution tool. It processes the log file created by the logging tool and produces in-memory Python dictionary representations for every unique process created while running a malicious program, including their respective child processes. The Neo4j graphing database management system ultimately visualizes in-memory dictionary objects via the Neo4j Python drive.

5.4. Using CloudMonitor to Detect Remote Computer Worm Attacks

The computer worm technique utilized in this experiment can infect all computers in a network after infecting a single computer [84]. During the assault, the attacker first accesses a computer in the network but with no awareness of the user and then transmits all the required files and scripts needed to infect all nodes within the consumer network [84]. Once the initial stage has been performed, the attack will detect all the consumer's neighbor IP addresses within the network and deliver the attack payload to the "/tmp" directory of all the systems through SCP with zero knowledge from the users. Then, from the node to which the adversary initially permits access, the SSH brute force tool initiates SSH attack sessions against each node [85]. Installing xHdra, Ncrack, and Patator tools to run

the attack on victim nodes without the users' awareness [85], decompressing the attack payload, compiling the attack code, and executing the attack. The goal here is to track the attack processes within the log files and generate trace graphs. Sections 5.3.1 and 5.3.2 cover the implementation details of SSH attacks.

5.4.1. Consumer-Side Attack Analysis

CloudMonitor was used to study a computer worm assault from the consumer's perspective. During the experiment, a virtual computer network comprised of many virtual machines, including user virtual machines and OwnCloud acting as the cloud storage server configured with VirtualBox was utilized. To mimic the attack, the start and termination scripts have been set up on the OwnCloud server so that the Netlink-based logger will be initiated just before the attack's execution. Within the execution of the assault on the user's virtual machines, a Netlink-based logging application tracks all processes initiated during the period of the attack, encompassing both attack-related actions and background activities. After the attack is complete, a signal is sent to the consumer side by the start and termination script to end the Netlink-based logger on the virtual PC of the victim. The Netlink-based logger then completes the process of logging 30 s after receiving the termination signal. As a result of the attack, after the logging process is finished, a log file named "log.txt" will be stored in the /Desktop/logger directory, along with a screenshot file placed on the user's desktop. The log file is then sent to a visualization tool to analyze the various processes and their associations utilizing a process trace graph.

This consumer-side visualization tool is comprised of several processes. During the execution of a computer worm attack, for instance, the SCP process acts as the entry process of the entire attack scenario, in which the attacker injects the necessary files and scripts as a zip payload to the target machine in the network. The SCP process originates from a background process operating within the Linux environment (host machine), ultimately evolving into the parent process. When the attacker initiates a SCP communication protocol with the user machines, a SCP process is created. This process navigates to the /tmp directory, the location used to deposit the "screengrab.zip" payload, which then is used by SSH processes created during the attack time frame to execute the attack.

During the execution of the attack, SSH processes are generated, with each SSH process spawning three offspring processes. Each SSH parent process spawns a child process that modifies its directory to /tmp, the directory from where the payload was initially injected onto the consumer machine. The second process, which originated directly from the SSH parent process, unzips the payload and subsequently initiates another process to perform an update of the Linux package repository using "sudo aptget update." which will eventually spawn additional child processes to update all existing Linux packages installed on the host system. After all packages have been updated successfully by the parent and child processes, the SSH processes spawn further children processes to run "sudo apt-get install libx11-dev" and "sudo apt-get install libpng-dev" (for capturing screenshots and saving them in ".png" format on the Linux file system respectively, typically need packages).

Attack processes launched during attack execution can be traced back to one of SSH's parent processes. Some of the tasks undertaken by these processes include updating repository packages, installing dependencies required to execute the attack, compiling the computer worm source code to fit into the consumer network, etc. Numerous variables, including the operating system, kernel version, and compiler version, affect the number of attack-related processes created.

The computer worm attack process is responsible for running the attack binary on the user's system when the attack process created from the attacker's SSH session completes the compilation of the computer worm source code. Since the source code of the attack was developed from a single-process, single-threaded execution in mind, only one attack is made during the attack's execution, allowing the attacker to run this attack on other machines.

5.4.2. Provider-Side Attack Analysis

The CloudMonitor analysis tool was used to perform a provider-side analysis of a computer worm attack from the cloud service's point of view. Similarly, to the consumer-side analysis scenario, networked virtual server machines within the VirtualBox environment [86] were utilized. While analyzing the provider's side, an Intel Pin-based logging tool was directly attached to the attack's shell script. This allowed for the capturing and logging of the processes generated throughout its execution. Upon successfully concluding and logging the assault, the Intel Pin-based logging tool creates a log file titled "processtrace.txt" in the /Desktop/screen-grab directory of the cloud storage system. Then, this log file is put into the CloudMonitor's visualization tool to determine the relationship between the processes and visualize how these processes interact with each other.

The provider's side perspective visualization tool is comprised of two separate processes, namely entry and attack. The entry process marks the beginning of the assault. Once the attack script has been started from the Linux terminal, the parent process starts a child process to execute it. On the attack side, those processes are created by the entry process. These attack processes can be traced back to the entry process itself, which is an instance of "sshpass4". The sshpass instance operating process then launches a child process to create an SCP communications channel with the victim's computer to inject the attack payload.

After injecting the payload is complete, the attack process starts another child process to launch a second instance of sshpass. This instance of sshpass executing on the attack process will also create a child process to create a communication channel of the SSH with the consumer computer. After completing the tasks of the SSH session, a second SSH session is initiated to clean the pre-compiled code. Following this, the attack source code is recompiled to match the requirements of the target machines, and then executing attack binary is remotely executed via the SSH session.

5.4.3. Performance Analysis Results

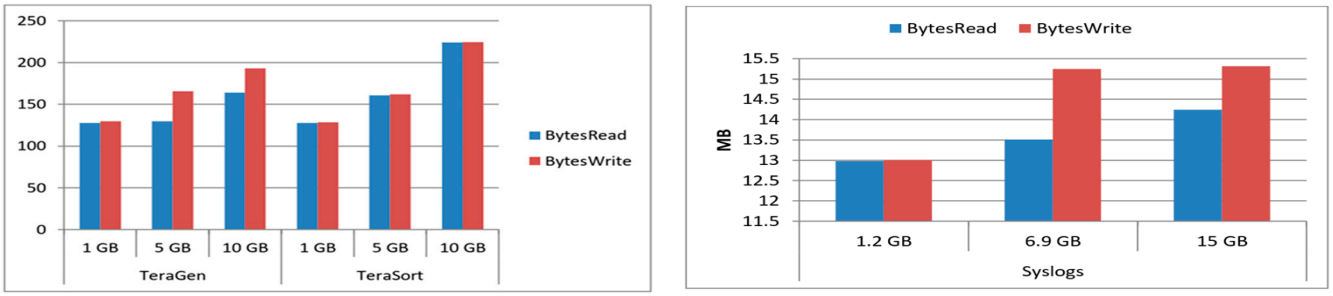
The performance evaluation of CloudMonitor involves several experiments (such as input/output performance, execution time, and resource utilization) at both the component and system level. At the component level, the focus is on the CloudMonitor components responsible for processing streaming data. While at the system-level, experiments assess the overall performance of CloudMonitor within the NiFi cluster. Apache NiFi is a user-friendly, robust, dependable data processing, and distribution system [86].

Customized reporting activities within NiFi facilitate metrics collection, measured over a rolling window of five minutes. Key metrics, including BytesRead and BytesWrite, are employed to assess input/output (I/O) performance, while Total Task Duration Seconds approximates execution time. Grafana, an open-source web application, aids in interactive analytics and visualization when connected to supported data sources [87].

Two distinct experiments are conducted to evaluate each system. The first involves assessing log volumes and data volumes using TeraGen and TeraSort benchmarks for three files of varying sizes (1 GB, 5 GB, and 10 GB) [88]. The second experiment employs various workloads of Syslog and DataNode log data with three files of different sizes (1.2 GB, 6.9 GB, and 15 GB) [89].

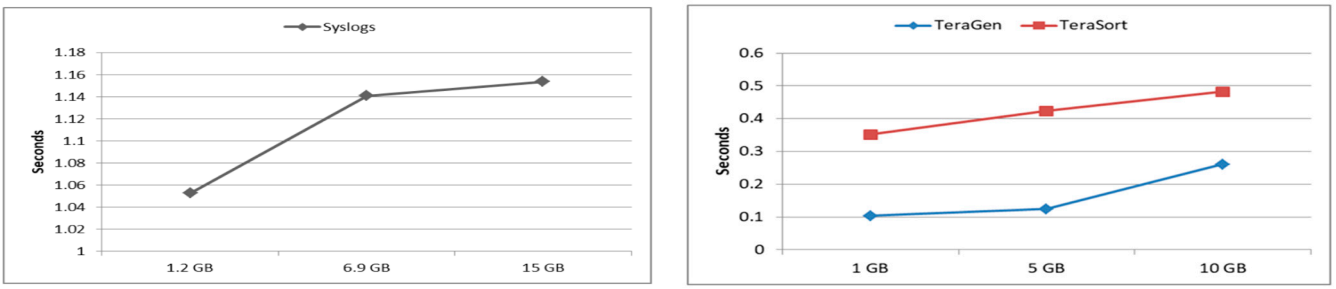
Figure 12 illustrates that performance scales proportionally with the size of the data, indicating that the analyzed data's size does not significantly impact performance. The performance is more dependent on the overall data volume, showing effectiveness within the range of 127 to 224 MB.

Figure 13 depicts the execution time for various data loads. The execution time increases nonlinearly as the volume of data increases. The primary reason stems from security checks that require processing time. In both instances, the system's execution time does not exceed one second. The experiment also measures the CPU usage and memory utilization of the machine. NiFi reporting tasks do not provide monitoring of resource consumption.



(**a**) (**b**)

Figure 12. Performance scales proportionally with data size, demonstrating effectiveness across a range of from 127 to 224 MB. Examination of TeraGen and TeraSort benchmarks (**a**) and workloads of Syslog and DataNode log data (**b**).



(**a**) (**b**)

Figure 13. Non-linear execution time increases with data volume due to security checks, while system performance remains under one second (Syslogs and data generated execution time (**a**) and TeraGen and TeraSort execution time (**b**)).

Accordingly, a monitoring component is employed as a collection of NiFi processors. The monitoring component uses the NiFi API to retrieve the CloudMonitor system diagnostic report. This report includes heap utilization and processor load average measurements. The monitoring component then uploads the refined metrics to Grafana via the AMS API [90].

Figure 14 depicts the average processor use of our system validation experiments and shows the proportion of heap memory consumed by CloudMonitor, respectively. As observed, usage changes throughout the duration of the experiment. The average utilization of the processor does not exceed 15% and averages roughly 8%. Similarly, the average memory usage is 37%, while the maximum is 46%. The average used heap memory is 3.7 GB, with a peak usage of 4.6 GB and the 95th percentile at 4.2 GB. According to the reported data in [90], the CloudMonitor system comparatively demonstrates an effective CPU and memory consumption profile (up to 35% in the worst-case scenarios).

5.4.4. Scalability in Perspective

Notably, the fundamental premise for cloud users (i.e., consumers' point of view) is usually stated in a service level agreement (SLA). In regard to scalability (more users, more data more data transfers), the assumption is that the CSP will spin up more VMs as needed. Naturally, there are real limits especially concerning network bandwidth, and concerning time and space in a commercial cloud (e.g., AWS). However, for our purposes, we have assumed that time and space are unlimited. The question then becomes, what is the relationship between a small-scale experimental cloud environment and one that is of industrial strength? We have ignored the network bandwidth aspect and assumed that the CloudMonitor DFT and Monitoring costs are linear with respect to the amount of data being moved (uploaded, moved, downloaded). Therefore, the relationship between Users: Applications: and Dataflows is roughly 1:1:1 meaning that the cost is UxAxD. Where

the number of users spawning data-intense applications that are consuming/producing data is multiplicative. However, any given SLA may differ, usually one finds that greater volumes offer greater discounts in real dollars. Whereas the costs and time may result in just the opposite depending on the specific hardware/server architecture. Our experiments are premised on this assumption. Most cloud environments are demand-driven, and thus the results are fundamental to the assumption of having unlimited resources. Thus, the relationship between the tracked data size and space/time cost increases as the fundamental unit of memory becomes smaller down to 8bits. Any smaller tracking unit would generally become too time/space limiting.

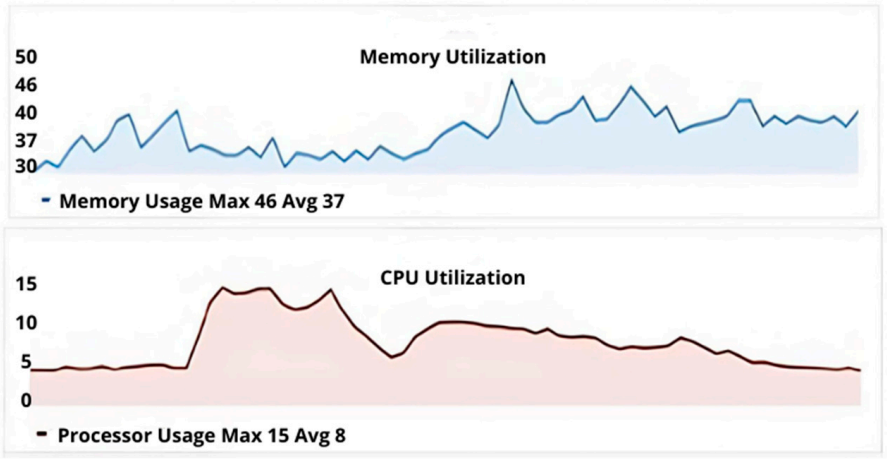


Figure 14. Average processor and heap memory utilization during system validation experiments.

6. Discussion

The comparative analysis of existing IFT tools emphasizes the need for mitigating data leakage in complex cloud systems. Traditional methods impose significant overhead on CSPs and management activities, prompting the exploration of alternatives such as IFT. By augmenting consumer data subsets with security tags and deploying a network of monitors, IFT facilitates the detection and prevention of data leaks among cloud tenants. This approach not only ensures data isolation but also minimizes the need for extensive configuration changes, thereby providing consumers with reassurance regarding the security of their data.

The CloudMonitor framework, as outlined in the preceding sections, presents a comprehensive framework for preserving data confidentiality and integrity in cloud services. Experimental evaluation of the framework's performance, both with and without IFT integration, reveals minimal overhead, particularly when coupled with IFT. Core modules such as integrity detection and emulated execution contribute to real-time monitoring and enforcement of privacy policies that showcase the approach's effectiveness in safeguarding consumer data. The CloudMonitor's ability to reject activities that violate privacy policies in real-time and cause alarms to trigger for audit purposes further underscores its utility in maintaining cloud security.

Detailed examination of data leak prevention mechanisms within the CloudMonitor elucidates its robustness against potential attacks. Integrating IFT into the CloudMonitor's control flow enables comprehensive monitoring of both consumer-side and provider-side activities which facilitates timely detection and response to abnormal behavior. Experimental scenarios validate the CloudMonitor's performance under various attack and defense conditions highlighting its scalability and adaptability in dynamic cloud environments. However, it is essential to acknowledge that there are limitations here. Our research has strived to establish a foundational framework, primarily that focuses on predefined attack scenarios and performance metrics, which may overlook specific security threats. Fortunately, the CloudMonitor framework can be tailored to better understand the intricacies of

new threats as they pertain to IFT performance and protection. However, in this work, we have supplied these critical building blocks, namely (Sections 4.3, 4.4, 5.1 and 5.2) covering the (a) CloudMonitor use-case, (b) formulation for enforcing the rules of data isolation, (c) data tracking policy framework, and (d) a basis for managing confidential data flow and data leak prevention using the CloudMonitor framework.

7. Conclusions

In our comprehensive review of existing IFT tools, we examined several analogs to CloudMonitor that address data leakage in complex cloud systems. Traditional methods often burden CSPs with significant overhead and management activities that drive the exploration of alternatives like IFT. Our analysis, focused primarily on the consumer side, augmented consumer data subsets with security tags and deployed a network of monitors which facilitated the detection and prevention of data leaks among cloud tenants. However, CloudMonitor stands out as a comprehensive framework designed explicitly for preserving data confidentiality and integrity in cloud services. By integrating IFT into its core modules, such as integrity detection and emulated execution, CloudMonitor offers real-time monitoring and enforcement of privacy policies with minimal overhead. Notably, CloudMonitor's ability to reject activities violating privacy policies in real-time and trigger alarms for audit purposes underscores its utility in maintaining cloud security.

IFT introduces a novel architecture for information security in corporate settings. It focuses on managing critical data flow to and from the cloud to prevent leaks and monitor changes. Unlike previous efforts, IFT emphasizes achieving complete binary-level compatibility with prevalent cloud services.

Our approach starts by considering consumer organizations' perspectives and proposing a monitoring tool by using a thin model approach positioned between consumer organizations and the cloud. This tool serves as a foundational element for a comprehensive information security platform. IFT aligns with existing cloud storage services while enabling thorough monitoring of user data and its interactions with external entities.

Implementing dynamic taint analysis and policy fulfillment within the cloud service enhances security by monitoring data flow and enforcing end-to-end Data Loss Prevention (DLP) constraints. However, significant runtime overhead due to dynamic taint analysis hampers mainstream adoption. Our study introduces algorithmic solutions to mitigate performance costs, including native machine instruction-level monitoring and asynchronous taint tracking.

Combining these optimizations, IFT achieves a notable performance improvement compared to previous approaches, into a framework we call CloudMonitor. Despite some overhead, particularly in CPU-bound scenarios, user feedback indicates manageable impacts on productivity. Notably, IFT stands out as the only system with a functional interactive interface and effective whole-system byte-level taint tracking.

Semantic gaps between cloud storage services and consumer infrastructure pose challenges in distinguishing intentional and unintentional data transmissions. Practical examinations reveal false tainting instances and highlight the need for further research into taint-spreading dynamics. Standard IFT techniques may not fully align with legacy application binaries, necessitating application-level restructuring.

To address these challenges, we propose methods for identifying and mitigating taint label leakage channels. While our initial results show promise, additional research is required to extend these methodologies to other cloud services.

Despite these advancements, it is essential to acknowledge the challenges encountered along the way. The phenomenon of taint explosion poses a significant hurdle that necessitates ongoing research efforts to better understand its dynamics and develop effective mitigation strategies. Furthermore, the semantic gap between cloud storage services and consumer infrastructure presents a practical constraint that requires innovative solutions necessary to overcome said constraints.

8. Future Work

We have outlined key research directions, primarily focusing on improving dynamic taint analysis performance. Despite past concerns about speed, our findings show significant reductions in overhead, enabling real-time analysis in the cloud. However, addressing performance alone is not enough for widespread acceptance; other obstacles persist.

Our work lays the groundwork for future research, utilizing our CloudMonitor prototype for ongoing investigations. To enhance data security in cloud environments, developers can utilize blockchain for secure data transfers [91] and AI for breach detection [92]. Control-flow integrity techniques ensure authorized access to sensitive data while combining static and dynamic analysis improves information flow tracking [93].

Efficient resource utilization is crucial and will be achieved through cost-effective cloud usage, algorithm optimization, and a pay-per-use model [94]. Modular cloud platforms offer flexibility and adaptability that could minimize development and maintenance costs while meeting evolving demands from both CSPs and CSCs.

Author Contributions: Conceptualization, F.A. and M.A.; methodology, F.A. and M.A.; software, F.A.; validation, F.A., M.A. and F.T.S.; formal analysis, F.A. and M.A.; investigation, F.A.; resources, F.A.; data curation, F.A.; writing—original draft preparation, F.A. and F.T.S.; writing—review and editing, M.A. and F.T.S.; visualization, F.A.; supervision, F.T.S.; project administration, F.T.S.; funding acquisition, F.A. and F.T.S. All authors have read and agreed to the published version of the manuscript.

Funding: This project was funded by the Deanship of Scientific Research at Prince Sattam bin Abdulaziz University award number (PSAU/2024/R/1445).

Data Availability Statement: The data presented in this study are available on GitHub. Specifically, the Intel-Pin Tools used are available at the GitHub repository: https://github.com/chiro2001/arch-lab/blob/master/lab1/examples/inscount2.cpp (accessed on 9 February 2023). Additionally, the Syslog workload code can be found at: https://github.com/syslog-ng/syslog-ng#!/usr/bin/envbash (accessed on 9 February 2023), and the DataNode workload code is available at the following repository: https://github.com/chenseanxy/helm-hadoop-3/tree/master/image (accessed on 9 February 2023).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

Acronym	Meaning
ACL	Access Control Lists
Aeolus	A platform for building secure distributed applications using the decentralized information flow control model
API	Application Programming Interfaces
Asbestos	A system (Circa '05) designed to enhance security by enforcing IFT at the protection domain level within a computing environment.
BLS	Boneh Lynn Shacham, a cryptographic signature scheme allows users to verify a signer is authentic
BytesRead	Number of bytes read by the operation from the disk to the cache.
BytesWrite	Total bytes are written
CCD	Consumer Confidential Data
CFI	Control-flow integrity
INTEL-PIN	Dynamic binary instrumentation framework
SafeWeb	Service that is designed to help users identify malicious websites
ISP	Internet Service Provider
NiFi	Niagara Files provides a way to stream rapidly flowing data across systems for automated storage, management, and manipulation in real-time.
LIBDFT	Dynamic Data Flow Tracking
LMBench	Micro-benchmark suite designed to focus on basic building blocks of many common system apps
MAC	Mandatory Access Control
MB	Megabyte
Ncrack	High-speed network authentication cracking tool

CIA	Confidentiality, Integrity, and Availability
CIFT	Centralized Information Flow Tracking
CloudFence	A DFT framework for cloud-based applications.
CloudFilter	Enforces email security policies and helps assure legal and regulatory compliance across your organization for inbound email
CPU	Central Processing Unit
CSC	Cloud Service Consumers
CSF	Cloud Security Framework
CSP	Cloud Service Providers
DAC	Discretionary Access Control
DAC	Dynamic Access Control
DataNode	Type of node in a distributed file system
DBI	Dynamic Binary Instrumentation
DisTaint	Dynamic IFT system that protects privacy and detects data leaks
DFT	Data Flow Tracking
DIFT	Dynamic Information Flow Tracking
DEFcon	Defense Ready Condition
DLP	Data Leak Prevention
Secure-ComFlow	System that employs IFC to secure cloud environments.
DStar	Digital voice and data protocol specification for amateur radio
DynamoRIO	BSD-licensed dynamic binary instrumentation framework to develop dynamic program analysis tools
TeraGen	This MapReduce program generates large data sets to be sorted
IFT	Information Flow Control Kernel Module
Flume	Information Flow Tracking
GCP	The commercial product Google Cloud Platform
IaaS	Infrastructure as a Service as opposed to SaaS and PaaS
FlowK	Distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.
IMBench	Set of applications for intermittently powered devices (https://github.com/CMUAbstract/imbench, accessed on 9 February 2023).
NFV	Network Function Virtualization
IT	Information Technology
PaaS	Platform as a Service
Patator	Multi-purpose brute-forcer, with a modular design and a flexible usage.
PID	Process ID
PPID	Parent Process ID
QEMU	Quick Emulator is a free and open-source emulator
RBAC	Role-Based Access Control
SaaS	Software as a Service
IoT	Internet of Things
SCP	Secure Copy
SDN	Software-Defined Networking
DoS	Denial of Service
SLA	Service-Level Agreements
SOP	Same Origin Policy
SSH	Secure Shell
sshpass	SSH Password
STAB	Debugging data format for storing information about computer programs for use by symbolic and source-level debuggers.
Stackdrive	A cloud computing systems management service
TEMU	Provided by the BitBlaze infrastructure for dynamic binary analysis to perform whole-system dynamic taint analysis
TeraSort	Typical Map/Reduce job; Sorts 1TB data (or any other amount of data) as fast as possible
EC2	Elastic Compute Cloud
VCPU	Virtual Central Processing Unit
Virtual RAM	Virtual Random Access Memory
VM	Virtual Machine
xHdra	Graphical version of hydra, a parallelized login cracker that supports numerous attack protocols
C&P	Consumers and Providers

References

1. Wang, Z.; Wang, N.; Su, X.; Ge, S. An empirical study on business analytics affordances enhancing the management of cloud computing data security. *Int. J. Inf. Manag.* **2020**, *50*, 387–394. [CrossRef]
2. Daylami, N. The origin and construct of cloud computing. *Int. J. Acad. Bus. World* **2015**, *9*, 39–45.
3. Moussa, A.N.; Ithnin, N.; Zainal, A. CFaaS: Bilaterally agreed evidence collection. *J. Cloud Comput. Adv. Syst. Appl.* **2018**, *7*, 1–19. [CrossRef]
4. Garg, D.; Sidhu, J.; Rani, S. Improved TOPSIS: A multi-criteria decision making for research productivity in cloud security. *Comput. Stand. Interfaces* **2019**, *65*, 61–78. [CrossRef]
5. Moussa, A.N.; Ithnin, N.B.; Miaikil, O.A. Conceptual forensic readiness framework for infrastructure-as-a-service consumers. In Proceedings of the 2014 IEEE Conference on Systems, Process and Control (ICSPC 2014), Kuala Lumpur, Malaysia, 12–14 December 2014.

6. Kumar, R.; Goyal, R. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. *Comput. Sci. Rev.* **2019**, *33*, 1–48. [CrossRef]
7. Moussa, A.N.; Ithnin, N.; Almolhis, N.; Zainal, A. A Consumer-Oriented Cloud Forensic Process Model. In Proceedings of the IEEE 10th Control and System Graduate Research Colloquium (ICSGRC), Shah Alam, Malaysia, 2–3 August 2019.
8. Jakóbik, A. Stackelberg game modeling of Cloud security defending strategy in the case of information leaks and corruption. *Simul. Model. Pract. Theory* **2020**, *103*, 102071. [CrossRef]
9. Calzavara, S. *Security II-Same Origin Policy*; Universita Ca' Foscari Venezia: Venice, Italy, 2020.
10. Roth, S.; Barron, T.; Calzavara, S.; Nikiforakis, N.; Stock, B. Complex Security Policy? A Longitudinal Analysis of Deployed Content Security Policies. In Proceedings of the 27th Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 23–26 February 2020.
11. Dehoumon, N. Controlled-Environment Facility Resident Communications Employing Cross-Origin Resource Sharing (USPTO 10,581,871). U.S. Patent 10,581,871, 3 March 2020.
12. Jadidi, M.S.; Zaborski, M.; Kidney, B.; Anderson, J. CapExec: Towards Transparently-Sandboxed Services (Extended Version). *arXiv* **2019**, arXiv:1909.12282.
13. Wang, H.; Wang, C.; Cai, Y.; Zhou, Q. A high-level information flow tracking method for detecting information leakage. *Integration* **2019**, *69*, 393–399. [CrossRef]
14. Almolhis, N.; Alashjaee, A.M.; Duraibi, S.; Alqahtani, F.; Moussa, A.N. The Security Issues in IoT-Cloud: A Review. In Proceedings of the 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA), Langkawi, Malaysia, 28–29 February 2020.
15. Hou, H.; Yu, J.; Hao, R. Cloud storage auditing with deduplication supporting different security levels according to data popularity. *J. Netw. Comput. Appl.* **2019**, *134*, 26–39. [CrossRef]
16. Walia, M.K.; Halgamuge, M.N.; Hettikankanamage, N.D.; Bellamy, C. Cloud Computing Security Issues of Sensitive Data. In *Handbook of Research on the IoT, Cloud Computing, and Wireless Network Optimization*; IGI Global: Hershey, PA, USA, 2019; pp. 60–84.
17. King, N.J.; Raja, V.T. Protecting the privacy and security of sensitive customer data in the cloud. *Comput. Law Secur. Rep.* **2012**, *28*, 308–319. [CrossRef]
18. Alassafi, M.O.; Alharthi, A.; Walters, R.J.; Wills, G.B. A framework for critical security factors that influence the decision of cloud adoption by Saudi government agencies. *Telemat. Inform.* **2017**, *34*, 996–1010. [CrossRef]
19. Ramachandra, G.; Iftikhar, M.; Khan, F.A. A Comprehensive Survey on Security in Cloud Computing. *Procedia Comput. Sci.* **2017**, *110*, 465–472. [CrossRef]
20. Bowers, K.D.; Juels, A.; Oprea, A. HAIL: A high-availability and integrity layer for Cloud storage. In Proceedings of the 16th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 9–13 November 2009.
21. PS, L.R. *Google Cloud Platform Cookbook: Implement, Deploy, Maintain, and Migrate Applications on Google Cloud Platform*; Packet Publishing Ltd.: Birmingham, UK, 2018.
22. Barsoum, A.F.; Hasan, M.A. *Provable Possession and Replication of Data over Cloud Servers*; Centre for Applied Cryptographic Research (CACR), University of Waterloo: Waterloo, ON, Canada, 2010.
23. Juels, A.; Kaliski, B.S., Jr. PORs: Proofs of retrievability for large files. In Proceedings of the 14th ACM conference on Computer and Communications Security, New York, NY, USA, 2 November–31 October 2007.
24. Shacham, H.; Waters, B. Compact Proofs of Retrievability. *J. Cryptol.* **2013**, *26*, 442–483. [CrossRef]
25. Guo, W.; Qin, S.; Lu, J.; Gao, F.; Jin, Z.; Wen, Q. Improved Proofs of Retrievability and Replication for Data Availability in Cloud Storage. *Comput. J.* **2020**, *63*, 1216–1230. [CrossRef]
26. Chang, J.; Shao, B.; Ji, Y.; Xu, M.; Xue, R. Secure network coding from secure proof of retrievability. *Sci. China Inf. Sci.* **2021**, *64*, 1–2. [CrossRef]
27. Gritti, C. Publicly Verifiable Proofs of Data Replication and Retrievability for Cloud Storage. In Proceedings of the International Computer Symposium (ICS), Tainan, Taiwan, 17–19 December 2020.
28. Kumar, R.; Goyal, R. Top Threats to Cloud: A Three-Dimensional Model of Cloud Security Assurance. In *Computer Networks and Inventive Communication Technologies*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 683–705.
29. Shynu, P.G.; Nadesh, R.K.; Menon, V.G.; Venu, P.; Abbasi, M.; Khosravi, M. A secure data deduplication system for integrated cloud-edge networks. *J. Cloud Comput.* **2020**, *9*, 61.
30. Wang, R. Research on data security technology based on Cloud storage. *Procedia Eng.* **2017**, *174*, 1340–1355. [CrossRef]
31. Renuga, S.; Jagatheeshwari, S.S.K. Efficient Privacy-Preserving Data Sanitization over Cloud Using Optimal GSA Algorithm. *Comput. J.* **2018**, *61*, 1577–1588. [CrossRef]
32. Han, P.; Liu, C.; Cao, J.; Duan, S.; Pan, H.; Cao, Z.; Fang, B. CloudDLP: Transparent and Scalable Data Sanitization for Browser-Based Cloud Storage. *IEEE Access* **2020**, *8*, 68449–68459. [CrossRef]
33. John, N.P.; Bindu, V.R. An Optimal Sanitization Algorithm Based Secure Migration of Virtual Machines in Cloud Datacenters. *Indian J. Comput. Sci. Eng.* **2021**, *12*, 709–718. [CrossRef]
34. Pasquier, T.F.M.; Powles, J.E. Expressing and enforcing location requirements in the cloud using information flow control. In Proceedings of the 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ, USA, 9–13 March 2015.
35. Dontov, D.; Klymenko, M. Decentralized Access Control for Cloud Services. U.S. Patent 16/183,575, 2019.
36. Han, L.C.; Susilo, W.; Huang, X. Fine-grained Information Flow Tracking using attributes. *Inf. Sci.* **2019**, *484*, 167–182. [CrossRef]

37. Gollamudi, A.; Chong, S.; Arden, O. Information Flow Tracking for distributed trusted execution environments. In Proceedings of the IEEE 32nd Computer Security Foundations Symposium (CSF), Hoboken, NJ, USA, 25–28 June 2019.
38. Chou, S.-C. An agent-based inter-application information flow control model. *J. Syst. Softw.* **2005**, *75*, 179–187. [CrossRef]
39. Bacon, J.; Eyers, D.; Pasquier, T.F.M.; Singh, J.; Papagiannis, I.; Pietzuch, P. Information Flow Control for Secure Cloud Computing. *IEEE Etransactions Netw. Serv. Manag.* **2014**, *11*, 76–89. [CrossRef]
40. Niu, B.; Tan, G. Efficient user-space Information Flow Tracking. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, New York, NY, USA, 8–10 May 2013.
41. Alpernas, K.; Flanagan, C.; Fouladi, S.; Ryzhyk, L.; Sagiv, M.; Schmitz, T.; Winstein, K. Secure serverless computing using dynamic information flow control. *Proc. ACM Program. Lang.* **2018**, *2*, 118. [CrossRef]
42. Yuan, J.; Qiang, W.; Jin, H.; Zou, D. CloudTaint: An elastic taint tracking framework for malware detection in the cloud. *J. Supercomput.* **2014**, *70*, 1433–1450. [CrossRef]
43. Liu, F.; Tong, J.; Mao, J.; Bohn, R.; Messina, J.; Badger, L.; Leaf, D. NIST Cloud Computing Reference Architecture. *NIST Spec. Publ.* **2011**, *500*, 292.
44. Mell, P.; Grance, T. The NIST Definition of Cloud Computing. *Commun. ACM* **2010**, *53*, 50.
45. Chess, B.; West, J. Dynamic taint propagation: Finding vulnerabilities without attacking. *Inf. Secur. Tech. Rep.* **2008**, *13*, 33–39. [CrossRef]
46. Efstathopoulos, P.; Krohn, M.; VanDeBogart, S.; Frey, C.; Ziegler, D.; Kohler, E.; Mazieres, D.; Kashooek, F.; Morris, R. Labels and event processes in the Asbestos operating system. *ACM Trans. Comput. Syst.* **2007**, *25*, 3.
47. Papagiannis, I.; Pietzuch, P. Cloudfilter: Practical control of sensitive data propagation to the cloud. In Proceedings of the ACM Workshop on Cloud Computing Security Workshop, New York, NY, USA, 19 October 2012.
48. Zeldovich, N.; Boyd-Wickizer, S.; Kohler, E.; Mazieres, D. Making information flow explicit in HiStar. *Commun. ACM* **2011**, *54*, 93–101. [CrossRef]
49. Krohn, M.; Yip, A.; Brodsky, M.; Cliffer, N.; Kaashoek, M.F.; Kohler, E.; Morris, R. Information Flow Tracking for standard OS abstractions. *ACM SIGOPS Oper. Syst. Rev.* **2007**, *41*, 321–334. [CrossRef]
50. Wang, X.; Ma, H.; Yang, K.; Liang, H. An Uneven Distributed System for Dynamic Taint Analysis Framework. In Proceedings of the 2nd International Conference on Cyber Security and Cloud Computing, New York, NY, USA, 3–5 November 2015.
51. Pappas, V.; Kemerlis, V.P.; Zavou, A.; Polychronakis, M.; Keromytis, A.D. CloudFence: Data Flow Tracking as a Cloud Service. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Gros Islet, Saint Lucia, 23–25 October 2013.
52. Khurshid, A.; Khan, A.N.; Khan, F.G.; Ali, M.; Shuja, J.; Khan, A.U.R. Secure-CamFlow: A device-oriented security model to assist information flow control systems in cloud environments for IoTs. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e4729. [CrossRef]
53. Joshi, A.; Purohit, P.; Jain, R. A Simplified Rule Based Distributed Information Flow Control for Cloud Computing. *Int. J. Comput. Sci. Inf. Technol.* **2015**, *6*, 1408–1414.
54. Leuprecht, C.; Skillicorn, D.B.; Tait, V.E. Beyond the Castle Model of cyber-risk and cyber-security. *Gov. Inf. Q.* **2016**, *33*, 250–257. [CrossRef]
55. Sun, Y.; Petracca, G.; Ge, X.; Jaeger, T. Pileus: Protecting user resources from vulnerable cloud services. In Proceedings of the 32nd Annual Conference on Computer Security Applications, New York, NY, USA, 5–8 December 2016.
56. Shyamasundar, R.K.; Kumar, N.N.; Rajarajan, M. Information-Flow Control for Building Security and Privacy Preserving Hybrid Clouds. In Proceedings of the 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Sydney, NSW, Australia, 12–14 December 2016.
57. Shahidinejad, A.; Nikoogoftar, E.; Ahsan, R. Software as a Service Placement in the Cloud Computing Using Genetic Algorithm. *Int. J. Ser. Eng. Sci.* **2020**, *6*, 22–33.
58. Hazelwood, K.; Kaeli, D.; Connors, D.; Reddi, V.J. Using Pin for Compiler and Computer Architecture Research and Education. 2007. Available online: https://www.intel.com/content/dam/develop/external/us/en/documents/pldi2007-pintutorial-256675.pdf (accessed on 9 February 2023).
59. Armknecht, F.; Bohli, J.M.; Karame, G.O.; Youssef, F. Transparent Data Deduplication in the Cloud. In Proceedings of the Conference on Computer and Communications Security, New York, NY, USA, 12–16 October 2015.
60. Min, S.L.; Pettit, R.; Puschner, P.; Ungerer, T. *Software Technologies for Embedded and Ubiquitous Systems*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2010.
61. Vachharajani, N.; Bridges, M.J.; Chang, J.; Rangan, R.; Ottoni, G.; Blome, J.A.; Reis, G.A.; Vachharajani, M.; August, D.I. RIFLE: An Architectural Framework for User-Centric Information-Flow Security. In Proceedings of the International Symposium on Microarchitecture, Los Alamitos, CA, USA, 4–8 December 2004.
62. Suh, G.E.; Lee, J.W.; Zhang, D.; Devadas, S. Secure program execution via dynamic information flow tracking. In Proceedings of the ASPLOS XI: Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, USA, 9–13 October 2004; Volume 39, pp. 85–96.
63. Zeldovich, N.; Boyd-Wickizer, S.; Mazieres, D. Securing Distributed Systems with Information Flow Tracking. In Proceedings of the NSDI '08: 5th USENIX Symposium on Networked Systems Design USENIX, San Francisco, CA, USA, 15 April 2008.

64. Cheng, W.; Ports, D.R.; Schultz, D.; Popic, V.; Blankstein, A.; Cowling, J.; Curtis, D.; Shrira, L.; Liskov, B. Abstractions for usable Information Flow Tracking in Aeolus. In Proceedings of the 2012 USENI Annual Technical Conference (USENI ATC 12), Boston, MA, USA, 13–15 June 2012.
65. Hosek, P.; Migliavacca, M.; Papagiannis, I.; Eyers, D.M.; Evans, D.; Shand, B.; Bacon, J.; Pietzuck, P. *SafeWeb: A Middleware for Securing Ruby-Based Web Applications*; Springer: Berlin/Heidelberg, Germany, 2011.
66. Waschke, M. *How Clouds Hold IT Together: Integrating Architecture with Cloud Deployment*, 1st ed.; Apress L.P.: Berkeley, CA, USA, 2015.
67. Geetha, P.; CR, R.R. SAMR: Optimal Workflow of VMs in Cloud Computing. In Proceedings of the International Conference on Recent Trends in Computing, Communication and Networking Technologies (ICRTCCNT'19), Tamil Nadu, India, 18–19 October 2019.
68. Dick, R.S.; Detmer, D.E.; Steen, E.B. *The Computer-Based Patient Record*; National Academies Press: Washington, DC, USA, 1997.
69. Ye, K. *Cloud Computing—CLOUD 2021*; Springer Nature: Berlin/Heidelberg, Germany, 2022.
70. Site, B.W. TEMU: The BitBlaze Dynamic Analysis Component. 2023. Available online: https://bitblaze.cs.berkeley.edu/temu.html (accessed on 9 February 2023).
71. Stone, G.R.; Bollinger, L.C. *National Security, Leaks and Freedom of the Press: The Pentagon Papers Fifty Years On*; Oxford University Press: Oxford, UK, 2021.
72. Ackerman, P. *Section 2: Industrial Cybersecurity—Security Monitoring*; Packt Publishing, Limited: Birmingham, UK, 2021.
73. Morovati, K.; Kadam, S.; Ghorbani, A. A network based document management model to prevent data extrusion. *Comput. Secur.* **2016**, *59*, 71–91. [CrossRef]
74. Rajole, V. *Causes of Data Breaches and Preventive Measures. Data Loss Prevention*; GRIN Verlag: Munchen, Germany, 2013.
75. Scribe, J.; Guan, J. Lecture 4: Dynamic Analysis and Fuzzing Presentation Logistics. 2019. Available online: https://www.cs.columbia.edu/~suman/dynamic_analysis_notes.pdf (accessed on 9 February 2023).
76. Levchenko, A.V.; Fyodorov, S.A. Dynamic Binary Instrumentation Tool for Data Locality Analysis. St. Petersburg State Polytechnical University Journal. Computer Science. *Telecommun. Control. Syst.* **2016**, *236*, 53–64.
77. Neira-Ayuso, P.; Gasca, R.M.; Lefevre, L. Communicating between the kernel and user-space in Linux using Netlink sockets. *Softw. Pract. Exp.* **2010**, *40*, 797–810. [CrossRef]
78. Dwivedi, H. *Implementing SSH*; John Wiley & Sons: Hoboken, NJ, USA, 2003.
79. Garfinkel, S.; Spafford, G. *Web Security, Privacy & Commerce*, 2nd ed.; O'Reilly Media, Incorporated: Newton, MA, USA, 2001.
80. Diogenes, Y.; Ozkaya, E. *Cybersecurity—Attack and Defense Strategies*; Packt Publishing Ltd.: Birmingham, UK, 2022.
81. Handbook, L. How to Find Process ID (PID and PPID) in Linux. 2022. Available online: https://linuxhandbook.com/find-process-id/ (accessed on 9 February 2023).
82. Manual, D.M.-C. Neo4j Graph Data Platform. 2023. Available online: https://neo4j.com/docs/operations-manual/current/database-administration/ (accessed on 9 February 2023).
83. IBM. What Are NoSQL Databases? 2023. Available online: www.ibm.com/topics/nosql-databases (accessed on 9 February 2023).
84. Ochieng, N.; Mwangi, W.; Ateya, I. A Tour of the Computer Worm Detection Space. *Int. J. Comput. Appl.* **2014**, *104*, 29–33. [CrossRef]
85. GoLinuxCloud. Automated SSH Brute Force Attack [4 Methods]. 2023. Available online: www.golinuxcloud.com/ssh-brute-force-attack/ (accessed on 14 February 2023).
86. Apache, N. NiFi System Administrator's Guide. 2023. Available online: https://nifi.apache.org/docs/nifi-docs/html/administration-guide.html (accessed on 9 February 2023).
87. Grafana. Data Analytics and Interactive Visualization. 2022. Available online: www.stackscale.com/blog/grafana/ (accessed on 28 February 2023).
88. IBM. TeraSort Benchmark. 2023. Available online: www.ibm.com/docs/en/spectrum-symphony/7.2.1?topic=mapreduce-terasort-benchmark (accessed on 9 February 2023).
89. Monitor, A. guywi-ms. Collect Syslog Data Sources with the Log Analytics Agent in Azure Monitor—Azure Monitor. 2023. Available online: https://learn.microsoft.com/en-us/azure/azure-monitor/agents/data-sources-syslog (accessed on 9 February 2023).
90. Labs, G. Ambari Metrics Plugin for Grafana. 2023. Available online: https://grafana.com/grafana/plugins/praj-ams-datasource/ (accessed on 9 February 2023).
91. Burgwinkel, D. Blockchain Technology. In *Blockchains-wichtige Fragen aus IT-Sicht*; De Gruyter: Berlin, Germany, 2017; pp. 123–148.
92. Winston, P.H. *Artificial Intelligence*; Addison-Wesley: Boston, MA, USA, 2019.
93. Wang, Y.; Li, Q.; Chen, Z.; Zhang, P.; Zhang, G.; Shi, Z. BCI-CFI: A context-sensitive control-flow integrity method based on branch correlation integrity. *Inf. Softw. Technol.* **2021**, *136*, 106572. [CrossRef]
94. Petrosian, L.G.; Ambartsumian, V.A. *Static and Dynamic Analysis of Engineering Structures*; John Wiley & Sons: Hoboken, NJ, USA, 2020.