Measuring Attack Observability in Cloud Telemetry Logs: A Cross-Platform Analysis

Mary Grace Dhooghe*, Minkyung Park*, Junghwan Rhee†, Yung Ryn Choe‡, Chung Hwan Kim*
*University of Texas at Dallas, †University of Central Oklahoma, ‡Sandia National Laboratories
{mary.kozuch, minkyung.park, chungkim}@utdallas.edu, jrhee2@uco.edu, yrchoe@sandia.gov

***Abstract*—With unparalleled flexibility in management and resource scaling, cloud computing is widely adopted and growing. Cyber threats targeting cloud instances require security solutions as much as physical infrastructures. Although traditional host-based and network-based security tools can be deployed to detect malicious activities within a cloud instance, they are not suitable for monitoring activities specific to cloud environments such as cloud resource management and authentication.**

In this paper, we systematically investigate the effectiveness of cloud telemetry logs, for cybersecurity monitoring and defense. While cloud infrastructures provide telemetry logs to help cloud users and administrators monitor various activities such as performance or resource management, there remains limited understanding of the extent to which these logs capture evidence of attacks, the types of attacks they can reveal, and their effectiveness across platforms. To address this gap, we conduct 35 attacks categorized under the MITRE ATT&CK framework on three major cloud infrastructures, and collect telemetry logs under both control and attack scenarios. Through differential analysis, we identify which log fields are impacted by specific attacks, assess platform-level consistency, and characterize gaps in log observability. Our new findings show that default telemetry configurations reflect 45∼48% of the attacks, while enabling extended logging increases this to 65∼88%, depending on the platform. We also evaluate the cost-effectiveness and performance overhead of telemetry logs, demonstrating minimal deployment costs to enable the extended logging.

***Index Terms*—Cloud Platforms, Telemetry Logs, Security**

I. Introduction

Cloud computing provides businesses with unparalleled flexibility in managing IT resources and infrastructures, which has led to a growing reliance on this technology across diverse industries [16], [11]. The increasing utility and criticality of cloud computing introduces the evolving threat landscape. Cloud instances have become a target of traditional threats [40] such as malware injection [35], hijacking execution flow [26], *etc*. Besides these, new forms of attacks specifically targeting cloud environments such as abusing cloud management services (*e.g*., Cloud Administration Command in [29]) as well as classic attacks reengineered to attack virtualized cloud resources (*e.g*., Data Destruction) have emerged.

Detection of cloud instance attacks hinges on comprehensive and timely data availability. Many attack detection and forensic analysis methods depend on key contextual information, such as the actor, action, timestamp, and affected resources. *Cloud telemetry logs*, cloud vendor-specific collections of event records, capture this salient contextual information. In this study, we evaluate their effectiveness as a key data source for attack investigations.

Security professionals often use log-based security solutions like Host-based Intrusion Detection Systems (HIDS) [42], [18], [23], [6] for traditional on-premise instances, informed by multiple types of monitoring information such as system-level logs, system call sequences [42], [6], call stacks [18], and system call arguments [23], to detect and respond to threats. These methods can also be applied to Infrastructure-as-a-Service (IaaS) resources (*e.g*., Virtual Machines) where system log information is available inside a deployed instance. However, the virtualized, distributed, and heterogeneous nature of cloud environments poses challenges to collecting these data sources. This is especially evident in the case of Platform-as-a-Service (PaaS) or Software-as-a-Service (SaaS) resources [15], which represent a large portion of cloud providers services (*e.g*. AWS S3), as access to such information is often unavailable where operating systems, middleware, or runtimes are virtualized. Furthermore, actions which exploit a cloud subscription configuration (*e.g*. Identity and Access Management (IAM)), potentially impacting all services within a compromised cloud subscription, can only be observable, if at all, through cloud-provided data.

We observe that *cloud telemetry logs* contain the types of information necessary for security monitoring. These logs are designed to offer insights into security, performance, accounting, pricing, and deployment requirements. They typically include details such as user identifiers, application activities, network activities, and accessed resources [36], [28], [12], [13], [14]. Since cloud telemetry logs capture resource-specific features using cloud system instrumentation code, they can gather more detailed action information. For instance, access to object-level storage in database services is recorded. As certain features are enabled by default, and others can be selectively activated, they provide cost-effective monitoring.

While cloud providers offer subscription-based security services (e.g., AWS GuardDuty, Azure Microsoft Defender, GCP Security Command Center), their implementations are often black-boxed, making their limitations difficult to assess, and their adoption remains limited [4]. Some of these services incorporate cloud telemetry logs as input [38]; however, the lack of detailed information about which telemetry logs are utilized and how they are leveraged positions this paper as a complementary contribution to existing cloud security. Specifically, the correlation between individual telemetry events and

concrete attack behaviors, as well as the overall effectiveness of cloud telemetry logs in revealing forensic evidence of attacks, remains insufficiently explored, thereby motivating the necessity of this study.

In this paper, we explore the capacity of cloud telemetry logs for detecting attacks in cloud environments. *We conduct a novel systematic investigation to understand the types of information these logs capture, identify which attacks leave detectable traces, and analyze how attacks influence the log content.* To this end, we collect and analyze telemetry logs from instances provided by three major cloud providers: Amazon Web Service (AWS), Google Cloud Platform (GCP), and Microsoft Azure. Our evaluation spans 35 different types of attacks across eleven attack categories in the MITRE ATT&CK framework [29], for which we collect logs both in the presence and absence of these attacks.

We perform systematic differential analysis to compare control logs with those affected by attacks to identify whether and how the telemetry events are influenced. Our analysis reveals that while telemetry logs exhibit consistent structure for each event type, the level of consistency and detail varies across platforms. We further characterize which attacks produce observable differences in log content, and identify which types of log fields are critical for detecting different classes of attacks. These findings uncover platform-specific logging gaps and highlight the potential of telemetry logs as a lightweight yet informative detection surface. Finally, we validate the feasibility of telemetry-based detection by implementing a deep learning–based detection mechanism guided by our empirical insights.

In summary, we make the following contributions:

- We undertake a comprehensive investigation across multiple cloud platforms, cloud resources, and attack scenarios to assess the availability and consistency of attack data in a variety of cloud telemetry logs.
- We develop a systematic approach to evaluate the presence and evidentiary content of attack-relevant data in cloud telemetry logs.
- Our differential analysis demonstrates that attacks have a measurable impact on cloud telemetry logs and identifies which categories of log configurations, attack types, and targeted resources are most visibly reflected in the telemetry data.

II. Background: Cloud Telemetry Logs

Cloud providers assign various IT resources to users, including computing power, storage capacity, network infrastructure, and application services. In cloud environments, monitoring systems play a crucial role in providing visibility into the usage, performance, and operational status of these resources, similar to traditional on-premises IT setups. Cloud monitoring systems enhance transparency by collecting telemetry data, enabling users to gain insights into resource utilization and system behavior. The collected logs are further analyzed and utilized to trigger actions, such as generating alerts or initiating automated responses.


```
{
  "eventVersion": "1.11",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "cloudtrail.amazonaws.com"
  },
  "eventTime": "2025-01-02T16:54:02Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "sourceIPAddress": "cloudtrail.amazonaws.com",
  "userAgent": "cloudtrail.amazonaws.com",
  "requestParameters": {
    "keySpec": "AES_256",
    "encryptionContext": { ... },
    "keyId": "arn:aws:kms:us-east-1:11111111:key/2222..."
  },
  "responseElements": null,
  "requestID": "33333333-3333-3333-3333-",
  "eventID": "44444444-4444-4444-4444-444444444444",
  "resources": [
    {
      "accountId": "11111111",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:us-east-1:11111111:key/2222..."
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "11111111",
  "eventCategory": "Management",
  ...
}
```


Listing 1: An example of a cloud telemetry log (CTL) entry from AWS is listed.

These monitoring solutions serve a variety of purposes, such as supporting accurate billing based on resource consumption, optimizing performance by diagnosing and identifying issues, and ensuring compliance with service-level agreements (SLAs). Given the significance of monitoring in cloud operations, most leading public cloud providers offer monitoring as part of their service portfolios. For instance, AWS provides CloudWatch, Microsoft Azure offers Azure Monitor, and Google Cloud delivers Cloud Monitoring.

The scope of data collection encompasses metrics, logs, traces, and events; however, for clarity and consistency, we use the term 'telemetry log' as a generalized reference without distinguishing between these specific types. The details regarding what information is gathered, how specific it is, and how it is structured often depend on the provider's architecture and the service being monitored.

The telemetry logs can be categorized based on their associated services or resources, such as IAM (Identity and Access Management), KMS (Key Management Service), virtual machines (*e.g.* AWS EC2), and others. Typically, each log entry includes the origin of an event (*e.g.* user identity and source IP address), metadata about the event (*e.g.* timestamp, associated resources, and event name), and detailed information specific to the event. The telemetry logs are provided in JSON format, with the structure and features of each entry varying depending on the log type, detail level, or log version. For instance, when a data key is generated by the KMS service, the log entry contains detailed key information, such as key specification and key identifier as shown in Listing 1.

When a user creates a cloud instance, certain logs are en-

abled by default to ensure baseline monitoring and compliance. Some of these logs are mandatory and cannot be turned off, as they are critical for system integrity and security (*e.g.*, audit logs or system activity logs). However, other logs can be enabled or disabled based on user preferences and operational needs. In addition to the default logs, users have the option to enable additional logs to gain deeper insights into specific activities. For instance, to monitor object-level API actions in AWS S3, such as DeleteObject, the user should explicitly activate object-level logging. These additional logs provide fine-grained visibility into resource usage and security events.

III. Threat Model and Attack Scope

We assess cloud telemetry logs to determine their evidential value for detecting attacks on cloud-hosted user systems. Our analysis is based on three assumptions about the integrity of the cloud platform and log management mechanism: (1) the attacker does not have access to the cloud provider's system, (2) the attacker has no ability to edit the logs at rest or in transit, and (3) the generated logs are accurate and retain their integrity throughout creation, transit, and storage.

To evaluate a range of attacks that generate system events detectable via cloud resource requests or manipulation of a system hosted on a cloud platform, we refer to the MITRE ATT&CK framework [29], a comprehensive knowledge base that categorises adversarial tactics and techniques observed in real-world cyber threats. This framework characterizes 227 attacks across 14 categories. The categorizations represent differing stages of an attack such as "Initial Access", "Persistence", and "Resource Development".

To assess the detectability of a broad range of attacks using cloud telemetry logs, we selected a representative subset from the MITRE ATT&CK framework according to the following criteria. Each attack was required to be *cloud-executable*, meaning it is performed in the victim's cloud environment; *reproducible*, able to be performed consistently on cloud resources relying on publicly available tools; and *ethically feasible*, excluding attacks that target the underlying cloud platform and focusing solely on user-controlled instances. Additionally, to maximize coverage, we prioritized *diversity* in attack behaviors, avoiding redundant or minimally varying patterns. All attacks chosen are outlined in Table I.

We additionally categorize the attacks into four distinct levels of attacker cloud instance permissions. For certain Credential Access attacks, such as Brute Force and the Exploitation of Unsecured Credentials, we assume the attacker has *no access* to the cloud instance, denoted as "A.1". In contrast, for some Initial Access attacks, we assume the attacker has *access to a cloud-hosted website*, denoted as "A.2". Additional Credential Access and Initial Access attacks, along with some Lateral Movement attacks, assume the attacker has constrained, low-level *user access to the cloud instance*, denoted as "A.3". For attacks under the Persistence, Command and Control, Collection, Execution, Impact, and Privilege Escalation categories, we assume the attacker possesses the *minimum privilege access required* to execute the malicious actions, denoted as "A.4". The attacker's initial access level per attack is presented in the "Access" column of Table I.

IV. Design of Cloud Telemetry Analysis

In this section, we outline our methodology for comparing logs generated during an attack with those from control scenarios under identical resource configurations but without malicious activity. Through comparative analysis of these logs, we identify distinctive patterns that could indicate malicious activity. Through this methodology, we evaluate the effectiveness of these logs for attack detection.

For our cloud telemetry analysis, we select the top major cloud service platforms for our targets: Google Cloud Platform (GCP), Microsoft Azure (Azure), and Amazon Web Services (AWS). In cases where the attacker has access to the cloud subscription, attacks are performed utilizing each individual platform's provided command-line interface. While we use the details and terms of these cloud platforms, our approach is generally applicable to any cloud service.

A. Scope of Cloud Resources Tested

The resources involved in each attack are chosen based on the specific requirements of the individual attack. This paper aims to analyze a diverse set of cloud-generated logs. Therefore, we prioritize, when possible, cloud-native resources that offer the *highest level of abstraction*. For instance, although cloud-hosted virtual machines include storage capabilities, for attacks involving data storage, such as "Data Destruction" or "Defacement", we instead use cloud storage services: GCP Cloud Storage, AWS S3, and Azure Storage.

It is important to note that MITRE ATT&CK attack techniques are defined broadly, and each can be implemented in multiple ways depending on context and environment. In this work, we select implementations that emphasize diversity and richness in log generation across platforms. However, alternative implementations are certainly valid and may yield different telemetry characteristics.

Table I provides a generic category of the relevant resources used for each attack as "Event Sub." (Event Subscription), "IAM" (Identity and Access Management), "KMS" (Key Management Service), "Secrets Mgr." (Secrets Manager), "SQL", "Storage", and "VM" (Virtual Machine). The platform specific resources used corresponding to the resource categories in Table I are provided in Table II.

The virtual machine instances are Linux-based unless otherwise noted in Table I. For AWS we utilize Amazon Linux machines rooted in Fedora and Ubuntu, GCP attacks use Debian 11 and Windows Server 2016, Azure's also implements Debian 11 as well as an Azure Core Windows 2022 machine.

B. Cloud Telemetry Log Collection

We collect logs from four cases, each repeated 10 times, to validate cloud log collection reliability and account for potential inconsistencies across iterations: two telemetry log configurations, *Default* and *Additional*, and two user action scenarios, *Control* and *Attack*.

Table I: Attack Categories and Description

Ref. ID	MITRE Cat.	MITRE Attack	Resource	Access	Payload Description
C.0	Collection	Archive Collected Data	Storage, KMS	A.4	Enforce Encryption on Bucket
C.1	Collection	Automated Collection	Storage	A.4	Automatically move files for exfiltration
C.2	Collection	Clipboard Data	Win./Ubuntu VM	A.4	Exfiltrate VM clipboard contents
C.3	Collection	Data Staged	Storage	A.4	Duplicate Storage files for exfiltration
CA.0	Credential Access	Brute Force	IAM	A.1	Guess Keys to bypass access controls
CA.1	Credential Access	Credentials from Password Stores	IAM, Secrets Mgr.	A.3	Steal key in Secret Mgr. as low-level user
CA.2	Credential Access	Steal App. Access Token	IAM, Storage,	A.3	Steal key from Storage as low-priority user
CA.3	Credential Access	Unsecured Credentials	IAM, Storage	A.1	Allow public access to stored Key
CAC.0	Command & Ctrl.	Ingress Tool Transfer	VM	A.4	Retrieve information about VM users
CAC.1	Command & Ctrl.	Remote Access Software	VM	A.4	Install remote desktop access on VM
E.0	Execution	Cloud Administration Command	VM, IAM	A.4	Assign user role from compromised VM
E.1	Execution	Command & Scripting Interpreter	VM, IAM	A.4	Manipulate user account from VM
E.2	Execution	User Execution	VM	A.4	Copy Malicious script to Cloud Hosted VM
Ex.0	Exfiltration	Automated Exfiltration	Storage, Event Sub.	A.4	Automatically Copy Data from Cloud
Ex.1	Exfiltration	Scheduled Transfer	Storage, Event Sub.	A.4	Scheduled Data Exfiltration from Cloud
I.0	Impact	Account Access Removal	IAM	A.4	Disable and Erase Service Account
I.1	Impact	Data Destruction	Storage	A.4	Destroy storage bucket and files
I.2	Impact	Data Encrypted for Impact	Storage, KMS	A.4	Compromise data through Encryption
I.3	Impact	Data Manipulation	Storage	A.4	Duplicate files for stealthy exfiltration
I.4	Impact	Defacement	Storage	A.4	Replace user file with malicious file
I.5	Impact	Disk Wipe	VM	A.4	Detach disk from VM, Delete Disk
I.6	Impact	Endpoint Denial of Service	SQL, VM (GCP)	A.2	Consistently transmit data to website
I.7	Impact	Inhibit System Recovery	Storage	A.4	Disable storage versioning and recovery
IA.0	Initial Access	Drive by Compromise	SQL, VM (GCP)	A.2	Exploit a SQL based website
IA.1	Initial Access	Exploit Public Facing App.	SQL, VM (GCP)	A.2	Use a website text box to inject JavaScript
IA.2	Initial Access[d]	Valid Accounts	IAM	A.3	Unauthorized Authenticate as User
LM.0	Lateral Mvmt.	Exploitation of Remote Services	VM	A.4	Anonymize SSH caller with Tor call to VM
LM.1	Lateral Mvmt.[d]	Use Alt. Auth. Material	IAM, Storage	A.3	Steal Data by impersonating high-priority user
P.0	Persistence	Account Manipulation	IAM	A.4	Escalate and Revoke Privileges
P.1	Persistence	Create Account	IAM	A.4	Create a malicious service account
P.2	Persistence	Event Triggered Execution	Storage, Event Sub.	A.4	Deploy an event trigger to automate actions
P.3	Persistence	Implant Internal Image	VM, Storage (GCP)	A.4	Import malicious VM image to cloud
P.4	Persistence	Scheduled Task/Job	Storage, Event Sub.	A.4	Deploy a timed trigger to automate actions
P.5	Persistence[d]	Traffic Signaling	VM	A.4	Signal vulnerable VM with Network Calls
PE.0	Privilege Esc.[d]	Access Token Manipulation	IAM	A.4	Generate Compromising Account Key

[d] indicates attack overlap with "Defense Evasion" category

Table II: Attack Resources Used (Acronyms per §IV-A)

Resource	GCP	AWS	Azure
Event Sub.	Eventarc	Lambda	Function & Logic Apps
IAM	IAM	IAM Users	AD Service Principals
KMS	KMS	KMS	Storage Encryption
Secrets Mgr.	Secrets Mgr.	Secrets Mgr.	Key Vault
SQL	SQL MySQL	RDS MySQL	SQL Serverless
Storage	Cloud Storage	S3	Storage
VM	Compute Engine	EC2	Virtual Machines

Default Log Configuration. The first log configuration, enabled by default, is intended for cloud users who do not actively review logs and have not modified the default settings, as explained in §II. Through testing, we gathered 12 types of default logs from GCP, including "Cloud Audit Activity" and "System Event". In comparison, AWS offers only one log type, "Management', and Azure provides just two: "Administrative" and "Policy".

Additional Log Configuration. The second log configuration extends the default configuration with additional supplemental logs enabled on each cloud platform, covering both audit-level and resource-specific data. Because logging mechanisms may evolve with platform updates, we provide a high-level overview of the categories of additional logs used in our analysis. Full details of the specific logs enabled in this study are provided in the appendix.

To support comprehensive and consistent analysis across platforms, we enable audit and diagnostic logs beyond the default configurations on GCP, AWS, and Azure. On GCP, this includes organization-level audit logs that capture administrative actions and data access events. On AWS, we configure CloudTrail to collect both management and data events across key services such as S3, Lambda, DynamoDB, along with SQL general logs. Azure provides a combination of subscription-level and resource-specific diagnostic settings, which we configure to capture both high-level service insights and fine-grained activity at the individual resource level.

Table III: Control Log Sets $U_{control}$ (Ex.: Attack PE.0)

delta	logName	methodName	...	resourceName
0	Activity	iam.createaccount	...	projects/A
65	Activity	iam.deleteaccount	...	serviceaccounts/***

Table IV: Single Attack Log Set (Ex.: Attack PE.0)

delta	logName	methodName	...	resourceName
0	Activity	iam.createaccount	...	projects/A
3	**DataAccess**	**iam.getiampolicy**	...	**serviceaccounts/*****
9	**Activity**	**iam.create[...]key**	...	**serviceaccounts/*****
70	Activity	iam.deleteaccount	...	serviceaccounts/***

The logs produced through these extended configurations offer deeper visibility into cloud activity and enable more detailed analysis of system telemetry. We refer to this enhanced logging setup as the "Additional Log Configuration". Throughout this paper, we compare this configuration with the "Default Log Configuration" described earlier.

Control Scenario. We implement a control scenario to isolate and collect baseline logs (*i.e.*, control logs) that are independent of attack payloads and to assess non-determinism across repeated executions. Control logs are generated by provisioning and de-provisioning the specific resources targeted by each attack type. Our objective is to determine whether attack-relevant evidence exists and to characterize its nature; thus, increasing the volume or complexity of control logs is unlikely to affect the presence or composition of this evidence. Table III shows a subset of these control logs, cleaned for readability.

Attack Scenario. To create the attack scenario, we incorporate attack payload actions between provisioning and de-provisioning. The logs retrieved from these scenarios are the attack logs and are exemplified in Table IV.

C. Attack Visibility in Telemetry Logs

We consider an attack to produce evidence in the telemetry logs, and thus be *visible*, if its execution results in log entries that differ from those generated under equivalent control conditions. This visibility demonstrates the feasibility of detecting attacks on a cloud subscription through telemetry log analysis.

Visibility Definition. Any attack that consistently demonstrates detectability in the cloud logs, indicated by payload-related attack log divergence from the collected control logs and verified as logically payload correlated through manual analysis, is considered "visible" in the cloud telemetry logs. An example of divergent attack logs, determining visibility, for attack PE.0 between Table III and Table IV is shown in the bolded second and third rows of Table IV.

Log Assessment. Attack log divergence is identified via complex set difference analysis, where logs that fail to merge with a maximal control log set on a per-entry basis are deemed divergent. Log comparison is done separately for each combination of attack (a), platform (p), and log configuration (c) through three steps: Cleaning, Merging, and Manual Verification.

Algorithm 1: Visibility Determination Algorithm

Data: $D \leftarrow [Collected\_Datasets]$, U, $is\_control$;
Result: U = Merged Datasets, $Visible[d] = d \setminus U$

```
for d in D do
    U_i ← U;
    Visible[d] ← ∅;
    // Compare Log Sets
    for entry in d do
        t ← entry['delta']
        U_ε ← {u ∈ U_i : |u['delta'] − t| ≤ ε} ;
        m ← merge(U_ε, entry) ;
        if m ≠ ∅ then
            i_m ← m_index[0];
            U_i ← U_i − U_i[idx_m, :];
        else
            if is_control then
                U ← U ∪ entry;
            else
                Visible[d] ← Visible[d] ∪ entry;
    U ← sort(U, 'delta');
```

Cleaning. The collection of logs for offline analysis presents several formatting challenges, including nested log formats, insignificant unique identifiers, and irregular multi-log events originating from the same user action. All logs provided as nested JSON objects are flattened and merged into a single dataset per attack and platform. Events are sorted by timestamp, then transformed into time deltas relative to the first log. Some telemetry logs represent a single user action or non-deterministic system events. In Azure logs, the "operationId" groups all logs from a single invocation; we retain only the initial log of each "operationId" to avoid redundancy. These logs contain event information such as "BeginRequest", or "EndRequest". However, these logs are not consistently generated; not all "EndRequest" logs have a corresponding "BeginRequest" and the converse is also true. This results in numerous inconsistencies in the Azure logs, making analysis and payload identification particularly challenging. Similarly, in AWS cloud logs, we remove log entries related to system activities that do not correspond to user actions. We discard columns containing only random log or event identifiers, as well as timestamp data previously captured by delta values. When random identifiers appear within otherwise consistent fields, only the variable portion is removed.

Merging. After standardizing the logs, we merge D, the set of 10 individual test run log datasets, to construct a single union log set, U_{control}. This set represents all possible control log entries for a given scenario, reducing the likelihood that inconsistencies in log generation are mistakenly attributed to the attack payload.

We obtain U_{control} with Algorithm 1 having input D, $U \leftarrow \emptyset$, and `is_control = True`. Each dataset of log entries, d, in D is iteratively compared against U_ϵ a subset of all previously merged log entries not already discovered in d, U_i, within an ϵ bounded time frame (Lines 5-6). If the inner join between the current log and this subset, U_ϵ, yields a non-empty set (Line 7), it indicates the presence of an identical log in U_i. We then remove the earliest matching log so that it may not be matched with any subsequent entries in d (Lines 8-9). If no match is found, this entry is added to U as a previously unseen control log (Line 12).

For each attack log dataset, any logs that cannot be merged with a log in the corresponding control dataset are tentatively attributed to the payload actions, indicating that the payload may be detectable through log analysis. When merging the attack datasets, the approach is similar to that of the control but differs in two places. First, we provide the unified dataset created by merging the control datasets, $U_{control}$, as the initial U dataset, rather than an empty dataset as in the control case, as well as setting `is_control = False`. Second, in the case where a matching entry is not found during the inner join, we add the entry to the $Visible$ data for the given attack test (Line 14) as we are now comparing the log entries of d with the control log entries. If the $Visible$ dataset for a given attack test is not empty, then we assign tentative visibility to be 'True' as this test shows payload dataset divergence from the control data, $U_{control}$.

Manual Verification. After merging, attack logs absent from control sets undergo quality control. For each attack, platform, and log configuration, logs flagged as divergent by the $Visible$ variable across the 10 sets are reviewed alongside $U_{control}$ to confirm that only relevant features influenced the merge, and against the payload definition to ensure the logs correspond to the payload. In most cases, all attack tests agree on visibility. When inconsistencies arise, we retain log lines present in more than half of an attack's tests and label an attack visible if over 80% of test cases meet the criterion. These thresholds balance inclusion and consistency, supported by §V-A, which shows low log variability with a higher visibility threshold for increased confidence.

V. Evaluation

We now present our findings on the logs corresponding to the attacks previously described in Table I, across the three major cloud platforms. Results are presented for both default log configurations (Def.) and additional log configurations (Add.). Our analysis provides data to address the following questions.

- **RQ1:** Do cloud telemetry logs provide (a) consistent and (b) relevant data for detecting malicious activity (§V-A-§V-B)?
- **RQ2:** What categories of (a) affected resources and (b) MITRE attacks are observable through cloud logs (§V-C)?
- **RQ3:** What specific information is found in the logs that can be analyzed in relation to malicious activity (§V-D)?

Table V: Log Consistency: Average values are reported for each platform's default control log sets, with $\#_{pa}$ representing the set of all log counts for each test under a given platform and attack.

	Default			Additional		
Platform	$avg(\#_{pa})$	$std(\#_{pa})$	m_{ratio}	$avg(\#_{pa})$	$std(\#_{pa})$	m_{ratio}
GCP	10.3686	0.0606	1.0672	29.1714	1.0657	1.0698
AWS	8.4667	0.6450	1.4361	24.2114	0.9773	1.3942
Azure	9.2387	2.7240	1.8409	11.8743	1.9760	1.6942

- **RQ4:** Do non-default logs provide additional insights into malicious activity, and what is the cost associated with collecting these extra logs (§V-E)?

A. Log Consistency

Data consistency is indicative of effective downstream log analysis. As described in §IV, to perform our visibility analysis we ran each control and payload scenario 10 times. We now present an analysis of log consistency for the control scenarios with data shown in Table V. We analyze the logs generated from the control scenario, as they serve as training data for downstream anomaly detection. We analyze each dataset after the cleaning step, described in §IV-C, as easily filtered logs do not accurately reflect the consistency of the analyzed logs.

We denote the number of log entries produced for a given control scenario pertaining to attack, a, and platform, p, as the set $\#_{pa}$, where each item in the set is the number of log entries for a given test for the relevant scenario. We measure numerical log consistency with the standard deviation in the number of log entries produced, $std(\#_{pa})$, for each control scenario.

To measure log data consistency, we examine the control log overlap between tests for each platform and attack quantified as $m_{ratio} = m_{pa}/(avg(\#_{pa}))$ where m_{pa} is the number of log entries in $U_{control}$ for each platform and attack. Across 10 test datasets, an m_{ratio} of 10 indicates no overlap, while a value of 1 corresponds to complete consistency in log content across runs.

Table V shows that all platforms are highly consistent, with $m_{ratio} < 2$. The GCP logs provide the most consistency in both number and content of log entries. AWS increases inconsistency but maintains a low coefficient of variation of $< 4\%$. Azure shows the most inconsistency, which is expected due to the non-deterministic nature of the Azure logs, discussed in §IV-C, where an unstable number of log entries are generated for a single event.

Answer to RQ1a: Across all platforms, cloud logs show an average coefficient of variation in log entry counts for each $\#_{pa}$ of 10.27% for default logs and 8.15% for additional logs. Combined with low $std(\#_{pa})$ and m_{ratio} values, this indicates consistent logging, with GCP being the most consistent and Azure the least.

B. Attack Visibility

We present Table VI with our findings on log visibility for each of the 35 attacks. For the GCP and AWS platforms with

Table VI: Visibility Per Attack, Platform, and Log Config.

Attack ID	GCP Def.	GCP Add.	AWS Def.	AWS Add.	Azure Def.	Azure Add.
C.0	✓	✓	✓	✓	✓	✓
C.1	X	X	X	✓	✓	✓
C.2	X	✓	X	X	✓	✓
C.3	X	✓	X	✓	X	✓
CA.0	X	X	X	X	X	X
CA.1	✓	✓	✓	✓	X	X
CA.2	X	✓	✓	✓	X	✓
CA.3	✓	✓	X	X	✓	✓
CAC.0	X	✓	X	X	X	X
CAC.1	X	✓	X	X	X	X
E.0	✓	✓	✓	✓	✓	✓
E.1	✓	✓	X	X	X	X
E.2	X	✓	X	X	X	X
Ex.0	✓	✓	✓	✓	✓	✓
Ex.1	✓	✓	✓	✓	✓	✓
I.0	X	X	X	X	✓	✓
I.1	✓	✓	X	✓	X	✓
I.2	✓	✓	✓	✓	✓	✓
I.3	X	✓	X	✓	X	✓
I.4	X	✓	X	✓	X	✓
I.5	✓	✓	✓	✓	✓	✓
I.6	X	✓	X	✓	X	X
I.7	X	✓	✓	✓	✓	✓
IA.0	X	✓	X	✓	X	X
IA.1	X	✓	X	✓	X	X
IA.2	X	X	✓	✓	X	X
LM.0	X	✓	X	X	X	X
LM.1	X	✓	✓	✓	X	✓
P.0	✓	✓	✓	✓	✓	✓
P.1	✓	✓	X	✓	✓	✓
P.2	✓	✓	✓	✓	✓	✓
P.3	✓	✓	✓	✓	✓	✓
P.4	✓	✓	✓	✓	✓	✓
P.5	X	✓	X	X	X	X
PE.0	✓	✓	✓	✓	✓	✓

the default logs enabled $\approx 45\%$ of the attacks tested were visible in the logs, and for Azure $\approx 48\%$ are visible. When additional logs are enabled, the overall visibility increased to, $\approx 88\%$, $\approx 71\%$, and $\approx 65\%$ for the GCP, AWS, and Azure platforms, respectively. We find strong visibility consistency across platforms within the default log configuration, indicating that relevant log information is similar across platforms. The highest visibility is found with the GCP additional logs, likely due to the inclusion of 'Data Read' logs, which capture both Cloud Storage actions and Virtual Machine access events, with the latter having low visibility with AWS and Azure.

Additionally, full-scale attacks often include multiple MITRE attack actions, increasing the probability that at least one attacker action will produce observable log evidence.

Answer to RQ1b: Our experiments show that between $\approx 42\%$ and $\approx 88\%$ of attacks exhibit evidence in the cloud logs. Table X further explores the relevant data supplied in this log evidence.

C. Categorical Visibility

Categorizing attacks by resource is appropriate, as many log types are intrinsically linked to the resource involved (§IV-B). Table VII shows the fraction of visible attacks grouped by resources for each platform and log configuration. As noted in Table I, many attacks involve multiple resources; therefore, a single attack is counted under all relevant resource categories.

Table VII: Visibility Per Resource

Resource	Atk. #	Default	Addtional
		GCP	
Storage	16	0.563	0.938
IAM	12	0.583	0.75
VM	13	0.308	1
SQL	3	0	1
Event Sub.	4	1	1
		AWS	
Storage	15	0.6	0.933
IAM	12	0.583	0.667
VM	10	0.3	0.3
SQL	3	0	1
Event Sub.	4	1	1
		Azure	
Storage	15	0.6	1
IAM	12	0.5	0.667
VM	10	0.4	0.4
SQL	3	0	0
Event Sub.	4	1	1

Event Subscription attacks are always visible, as they invoke cloud-hosted functions that generate extensive logs upon initialization. The visibility of the storage attacks is highly consistent across platforms and almost doubles in average visibility when the additional logs are implemented. This improvement occurs because file access events are not recorded by default on any platform and but can be explicitly enabled. Similar consistency across platforms is seen with the IAM attacks, which overlap frequently with the Storage attacks. VM attacks show low visibility except in GCP Additional Logs, where SSH requests can be captured; otherwise, VM activity is not transmitted, leaving many actions absent from the logs. None of the platforms collect SQL-specific logs by default. This can be enabled for forwarding to the telemetry logs in some capacity for select SQL implementations. In our testing, these logs were collected on GCP and AWS using MySQL. On Azure, where Serverless SQL was required due to automated authentication limitations, only SQL Error logs were collected.

Answer to RQ2a: We find high consistency in resource visibility across platforms. Over 50% visibility is achieved for all resources except VM and SQL, highlighting the potential for future work leveraging cloud telemetry logs to detect attacks involving Storage, IAM, and Event Subscription resources.

Table VIII groups attack visibility by MITRE category across platforms and log configurations. With the default log configuration, at least one attack is visible in each category, except Command and Control, which is not visible on any platform. Likewise, Initial Access and Lateral Movement are detectable only on AWS. Enabling additional logs increases visibility: Command and Control becomes observable on GCP, Lateral Movement is detectable across all platforms, and Initial Access shows improved visibility on both GCP and AWS. This

Table VIII: Visibility Per Mitre Category

		Default			Additional		
Mitre Category	**Atk. #**	GCP	AWS	Az.	GCP	AWS	Az.
Collection	4	0.25	0.25	0.75	0.75	0.75	1.00
Command & Ctrl.	2	0.00	0.00	0.00	1.00	0.00	0.00
Credential Access	4	0.50	0.50	0.25	0.75	0.50	0.50
Execution	3	0.67	0.33	0.33	1.00	0.33	0.33
Exfiltration	2	1.00	1.00	1.00	1.00	1.00	1.00
Impact	8	0.38	0.38	0.50	0.88	0.88	0.88
Initial Access	3	0.00	0.33	0.00	0.67	1.00	0.00
Lateral Movement	2	0.00	0.50	0.00	1.00	0.50	0.50
Persistence	6	0.83	0.67	0.83	1.00	0.83	0.83
Privilege Escalation	1	1.00	1.00	1.00	1.00	1.00	1.00

Table IX: Log Feature Information

	AWS	Azure	GCP
Action	3	8	2
Success	0	3	2
Caller	3	2	3
Event ID	1	4	0
Log ID	2	0	1
Log Type	3	1	2
Resource	2	8	5
Time	1	2	2
Total	15	28	17

behavior is consistent with the underlying attack mechanisms. Both Command and Control attacks involve sending malicious requests to cloud-hosted virtual machines which show much higher visibility in GCP Additional than in any other case as previously discussed.

Answer to RQ2b: In the default log scenario, the "Command and Control" category was undetectable on all platforms, and "Initial Access" and "Lateral Movement" attacks were undetectable on Azure and GCP. All other categories exhibited some visibility across all platforms. The degree of visibility observed highlights which categories should be prioritized in future research and security implementations involving cloud telemetry logs.

D. Log Features and Log Data Syntax Comparison

We analyze the consistent information available in cloud logs. Each cloud platform produces different telemetry logs, with variations based on factors such as event trigger, accessed resources, action type, and enabled log information. To facilitate a fair comparison, our analysis focuses solely on the default logs.

We begin by identifying the most common log features for each platform, comparing all datasets collected from both control and attack scenarios, and retaining only the features that appear in every dataset for a given platform. This results in 17 consistent features throughout the GCP logs, 57 in Azure, and 15 for AWS. We describe the actual occurrence on the cloud system as an "event", a single event can produce none, one, or multiple cloud log entries. We additionally define a "caller" as the user or resource that initiated the event.

Further analysis of the Azure log features shows that 23 of these features are nested in the "Claims" JSON web token, used to authenticate a user. We therefore consider these to be one singular feature as many of the values are duplicates of other features such as event or resource identifiers. We also exclude 7 additional features provided by Azure from our analysis due to duplication. This results in 28 unique Azure log features for analysis.

We categorize the collected features for each platform into broader information-based categories, shown in Table IX, based on cloud platform provided information and our collected logs [36], [28], [12], [13], [14]. We find that in all cases, all default logs analyzed provided information on the action taken, the event caller, an event or log identifier, the type of log, the resource accessed by the action or caller, and the time at which this event happened. We find that GCP only applies an identifier to the log entry, not the event, which reduces the ability to tie different events shown in the log together. Alternatively, Azure supplies multiple levels of event identifiers but does not provide log entry-specific identifiers. AWS provides both event and log entry identifiers, but contrary to GCP and Azure, does not consistently provide information on the success of actions.

To examine the attack information captured in cloud logs, we analyze the abnormal log entries for each scenario. We break down our visibility definition into 3 visibility categories: Full, Partial, and None. Where "Full" visibility requires that all malicious payload details are identifiable from the payload-generated logs; specifically, the action, actor, and affected resource must be logged. For example, PE.0, Access Token Manipulation, where the action taken was to create a service account key and the logs show the action "create-serviceaccountkey" and the specific service account used to call this action. "Partial" visibility applies when logs exist but lack sufficient detail to identify malicious characteristics, describing attacks where the exact actions taken were not clear or a portion of the malicious events were reported but not all actions taken created a log, such as in P.0, Account Manipulation, where the malicious action is to both grant and remove a service account role but the action reported in the logs simply shows "setIamPolicy". "None" visibility indicates that either no abnormal logs were discovered, or the abnormal logs do not clearly map to the payload actions.

To compare these two sources, we use a pretrained sentence-transformer model to generate embeddings for all observed commands and log events. Actions are clustered semantically grouping similar operations such as "create," "insert," and "enable," while keeping opposing actions like "create" versus "delete" separated. This allows the system to match each expected payload action to the most similar audit-log action, even when the names differ across platforms or abstractions. By testing each payload command for inclusion in the log events, we quantify whether each payload action is clearly reflected in the logs (full, partial, or no visibility).

This method provides a consistent cross-platform representation of actions, but it also introduces cases where platform-

Table X: Attack Information, where each column shows the "Abnormal / Total" where "Total" reports the average number of logs for each attack scenario and "Abnormal" the average number of abnormal logs, indicating an attack. We shade the cells *Dark Green*, *Medium Yellow*, and *Light Red* to indicate "Full", "Partial", and "None" visibility levels

	GCP		AWS		Azure	
	Def.	Add.	Def.	Add.	Def.	Add.
C.0	4 / 6	27 / 39	3 / 9	10 / 26	7 / 20	21 / 57
C.1	0 / 7	0 / 44	0 / 5	4 / 18	5 / 15	51 / 92
C.2	0 / 10	2 / 20	0 / 17	0 / 35	1 / 15	2 / 8
C.3	0 / 3	85 / 108	0 / 4	8 / 27	0 / 13	4 / 17
CA.0	0 / 0	0 / 0	0 / 0	0 / 2	0 / 0	0 / 0
CA.1	3 / 10	9 / 27	4 / 19	8 / 40	0 / 6	0 / 6
CA.2	0 / 9	1 / 27	2 / 9	3 / 24	0 / 13	1 / 15
CA.3	1 / 6	2 / 10	0 / 0	0 / 0	1 / 8	2 / 19
CAC.0	0 / 12	2 / 24	0 / 15	0 / 34	0 / 12	0 / 17
CAC.1	0 / 12	2 / 24	0 / 17	0 / 33	0 / 12	0 / 16
E.0	1 / 11	2 / 16	2 / 21	4 / 42	1 / 3	1 / 4
E.1	2 / 14	5 / 29	0 / 16	0 / 36	0 / 14	0 / 20
E.2	0 / 12	2 / 22	0 / 16	0 / 33	0 / 14	0 / 16
Ex.0	5 / 12	11 / 24	2 / 25	87 / 281	5 / 12	11 / 24
Ex.1	3 / 11	7 / 23	6 / 24	15 / 71	3 / 11	7 / 23
I.0	0 / 1	0 / 1	0 / 3	0 / 8	1 / 3	1 / 4
I.1	1 / 2	103 / 154	0 / 2	52 / 104	0 / 8	3 / 17
I.2	1 / 5	4 / 20	3 / 5	6 / 31	1 / 8	1 / 13
I.3	0 / 3	42 / 65	0 / 5	8 / 27	0 / 7	4 / 14
I.4	0 / 2	2 / 6	0 / 2	1 / 8	0 / 6	1 / 11
I.5	2 / 19	5 / 32	7 / 22	15 / 46	3 / 18	5 / 31
I.6	0 / 2	41 / 76	0 / 0	96 / 120	0 / 1	0 / 3
I.7	0 / 1	2 / 12	2 / 6	4 / 13	1 / 10	2 / 14
IA.0	0 / 2	4 / 37	0 / 0	6 / 30	0 / 1	0 / 3
IA.1	0 / 2	3 / 36	0 / 0	5 / 30	0 / 1	0 / 3
IA.2	0 / 0	0 / 0	1 / 2	2 / 4	0 / 1	0 / 1
LM.0	0 / 12	2 / 22	0 / 16	0 / 35	0 / 14	0 / 16
LM.1	0 / 13	3 / 29	1 / 8	3 / 24	0 / 15	1 / 16
P.0	2 / 2	5 / 5	2 / 3	4 / 8	2 / 4	2 / 5
P.1	2 / 4	2 / 4	0 / 1	4 / 4	1 / 2	1 / 3
P.2	9 / 168	17 / 229	1 / 12	6 / 35	14 / 21	1846 / 2465
P.3	315 / 333	684 / 733	22 / 33	40 / 68	3 / 30	1 / 13
P.4	380 / 414	551 / 613	6 / 16	6 / 18	5 / 15	10 / 33
P.5	0 / 12	4 / 26	0 / 16	0 / 33	0 / 12	0 / 16
PE.0	1 / 3	2 / 4	2 / 4	7 / 9	1 / 6	1 / 10

specific terminology may be misaligned. For example, Azure CLI commands may refer to file systems using the abbreviation 'fs', while the corresponding logs record these operations under the 'storageaccounts' service. To mitigate these inconsistencies, we manually supplied the encoder with a set of partial-string mappings so that semantically equivalent platform-specific terms are normalized before clustering.

This analysis is presented in Table X, where we also report the average number of logs and the average number of abnormal logs detected for each attack and log configuration. As expected, in both configurations, a higher number of abnormal logs often corresponds to the "Full" visibility label. Attacks targeting data in storage are generally fully visible. However, attacks that modify storage settings, such as I.7 (Inhibit System Recovery), or access files, like CA.2 (Steal Application Access Token), are only partially visible. Additionally, we find that nearly none of the attacks involving virtual machines are fully visible, as the abnormal logs only show access to the machine without detailing the actions performed on it.

Answer to RQ3: All default logs analyzed across platforms provided information on the action taken, event caller, type of log, resource accessed, event time, and an event or log identifier. Semantic analysis of logs compared with payload actions reveals that only about half of the visible attacks have all actions specifically shown in the log data.

E. Log Configuration Cost Comparison

Monetary Cost. Throughout our testing we run 10 test cases for each of the 35 attacks in both the control and attack scenarios. Overall, enabling the additional logs on all platforms resulted in extra storage and logging costs of less than $1. However, for larger systems, the additional logging and storage costs may incur noticeable costs. For example, when comparing default and additional log configurations on the Azure platform, the generated log data amounted to approximately 0.066 GB and 0.202 GB, respectively. This indicates that enabling the additional logs in Azure results in over 200% increase in storage costs. For reference, Azure currently provides pay-as-you-go pricing for Analytics Logs and Basic Logs, with prices of $2.99 per GB and $0.65 per GB respectively [5].

Performance Overhead. We additionally test the performance overhead of enabling additional logs with attack I.1, Data Destruction. This attack shows a large discrepancy in logs generated between the Default and Additional log conditions on all platforms and is therefore a good indicator of additional log performance overhead. We run this attack under both default and additional log conditions 3 times each, recording the total time the attack took to run for each log configuration and platform. We find through this analysis that the GCP, AWS, and Azure logs incur a -0.40%, 2.81%, and 1.28% performance overhead, respectively, when the additional logs are enabled as compared with the default logs. We also find that the standard deviations of the AWS and Azure platforms in the default scenario are 10.14 and 6.48 seconds, which is 4.48% and 1.13% of the mean, respectively. We find the performance overhead of implementing the additional logs to be insignificant.

Answer to RQ4: As shown throughout §V, collecting additional telemetry logs can increase attack visibility by up to 50 percent, seen on the GCP platform. In high-usage scenarios, these logs may incur monetary costs, but testing shows no noticeable performance impact.

F. Machine Learning Concept Validation

We provide a simple proof-of-concept with a machine learning-based anomaly detection mechanism employed to learn the typical log structure and content from each cloud platform and identify anomalous logs.

Architecture. We implement a Variational Autoencoder utilizing Long Short Term Memory (LSTM) layers. As discussed, each log consists of a nested JSON object which can be flattened into a single feature set where each feature label is the JSON key and the feature value is the JSON value. With each

log as a single instance, *i.e.* row, containing multiple features, *i.e.* columns, we can translate our logs into a two-dimensional time-ordered dataset.

Autoencoders are commonly used for anomaly detection in multi-dimensional datasets, examples include [34], [47], [3]. In an autoencoder system, two distinct neural networks are utilized: the encoder and the decoder. The encoder processes an input, X, compressing it into a lower-dimensional representation, z. This is then passed to the decoder, which reconstructs the original input by expanding z back to the original dimensionality, yielding the output X'. The reconstruction loss is computed as the mean squared error between X and X', quantifying the information loss during the dimensionality reduction and reconstruction process.

The system is trained in an unsupervised manner to capture essential features of control dataset, X_b. During the inference phase, any log entry in unseen dataset X_u with a reconstruction loss above a defined threshold is flagged as anomalous relative to the training dataset. In this manner, we detect individual log entries that may be anomalous, potentially indicating malicious behavior on the cloud system.

For our approach, we use a variational autoencoder (VAE), proposed in [21] and further explained in [22]. Unlike standard autoencoders, the encoder in a VAE maps the input to a probabilistic latent space rather than a deterministic one. The VAE approach is often used in place of a traditional autoencoder, such as in [3], due to its probability distribution learning, providing a greater understanding of sample probability and reducing data memorization.

The timing and sequence are crucial for tracing user actions and identifying anomalies. Due to this time-dependent nature, we employ an LSTM neural network architecture for both the encoder and decoder. The architecture of an LSTM is specially tuned to learn time series data and is commonly used in log-based anomaly detection [17].

Results. We evaluate the collected cloud logs for machine learning anomaly detection using the control logs collected in §V, split into training, validation, and testing as 50:20:30, respectively. To increase the training dataset size, training data is augmented with random noise at each training round. We train a VAE, as described previously, for each attack individually, then test the attack data and holdout control data to assess detection through anomaly analysis. For each platform and log configuration, Table XI reports the number of detected attacks and falsely detected control datasets, along with the average true positive and false positive detection rates.

We present our findings by distinguishing visible attacks from those without visibility, as shown in §V-A. It is expected that, when the attack is not visible in the log data, the attack is not detected by the anomaly detection system. We are able to detect 35% or more of the visible attacks in all scenarios. This relatively low number is not unexpected because, as shown in Table X, most of the attacks in the default configuration are visible in only 1 or 2 logs. Additionally, compared with the total number of logs collected, the amount that are truly connected to an attack event is relatively low. Although there are 35 attacks, the number of control attacks in each row is not 35. This is because, for some attacks, the control condition does not produce any logs and therefore no anomaly detection system could be trained.

Table XI: Anomaly detection results. 'T.P.' and 'F.P.' denote the true and false positive rates, respectively; 'Acc.' is the overall accuracy. 'Vis.', 'Inv.', and 'Ctrl.' refer to the number of detected Visible, Invisible, and Control attacks, shown as a fraction of the total attacks in each category.

Platform	Log	T. P.	F. P.	Acc.	Vis.	Inv.	Ctrl.
GCP	**Def.**	0.654	0.044	0.977	9/15	0/17	1/32
	Add.	0.294	0.105	0.861	16/30	0/2	4/32
AWS	**Def.**	0.409	0.080	0.909	8/16	0/13	2/30
	Add.	0.227	0.095	0.838	11/24	1/9	5/33
Azure	**Def.**	0.325	0.004	0.953	7/16	2/15	2/31
	Add.	0.408	0.103	0.868	12/23	1/11	4/34

Case Study: Web Application.

Through our testing, we discovered that AWS Cloudwatch has a currently implemented "Log Anomalies" function that is available for use with collected telemetry logs. However, this function is not very thoroughly documented nor is there hyperparameter tuning available to the user to assist with a deeper understanding of the architecture of this mechanism. AWS provides this description within the cloud web console, "Log Anomaly Detection uses Machine Learning to automatically monitor your logs and surface an Anomaly whenever unusual behavior such as a new ERROR message or a large increase in logs occurs. [...] Each anomaly is based on a Pattern, a shared text structure that recurses among your log fields." [37]

To evaluate whether this AWS function fully leverages the information we identified as available in the logs, we simulate a cloud subscription hosting a website with frequent SQL requests and occasional modifications to users, virtual machines, and storage. We run this scenario for 24 hours to create a control log set, sending all logs to AWS CloudWatch. We then train an AWS Anomaly Detector on the CloudWatch group and use the same log set to train a variational autoencoder as described in §V-F. Following this control log collection, we enable the trained AWS Cloudwatch anomaly detector and continue to run the relational database system with frequent SQL requests while simultaneously running each of our control and attack scenarios 3 times each for all 35 attacks. Through this test, we find that *the currently implemented AWS anomaly detection system does not detect any of the 35 attacks tested*. In contrast, our simple VAE implementation detects 10 of the 35 attacks while falsely labeling only 4 control log sets when trained on the control web application simulation logs. Based on this experiment as well as the brief description of the anomaly detection function provided by AWS, we postulate that this system is not designed to discover anomalies unrelated to ERROR messaging. Furthermore, although the description indicates that 'large increases in logs' can be labeled as anomalous, Attack I.1 is not detected despite generating over 100 logs.

VI. Related Work

Cloud attacks encompass both traditional attack methods targeting on-premise environments [40], [35], [26] and cloud-specific attack vectors. The latter includes vulnerabilities in management interfaces, network protocols, APIs, and other components unique to cloud architectures. Such attacks can lead to severe consequences, including unauthorized access to sensitive data, service disruptions, and financial losses. To detect these threats, log-based attack and anomaly detection techniques have been developed and studied extensively over the years. Due to the longstanding research interest in this area, there are numerous survey and review papers providing its comprehensive overviews [24], [33], [20], [10], [2], [9].

Most existing studies focus on system logs for detecting security threats and failures [32], [7], [31], [41], [8], [19], [45], [43], [46], [27], [30], [25]. These works primarily aim to improve detection accuracy by modeling patterns in system-generated events, often using deep learning techniques tailored for structured or unstructured log data such as OpenStack dataset and network traffic data. For example, DeepLog [17] and LogRobust [44] apply LSTM-based models to system logs to predict or embed log sequences for anomaly detection. Although these approaches demonstrate promising results, they depend on system-level logs that are often not readily accessible in public cloud environments.

Only a few works [1], [39] have explored the use of real-world cloud telemetry logs. Notably, prior research has focused exclusively on logs from Infrastructure as a Service (IaaS) resources, leaving a significant gap in the study of logs generated by Platform as a Service (PaaS) and Software as a Service (SaaS) environments, where provider-generated logs are especially valuable due to limited user access to system-level logs. Agrawal *et al*. [1] proposes a Principal Component Analysis for anomaly detection on Amazon Cloudwatch server metrics consisting of CPU utilization, memory utilization, and disk I/O data. However, their focus is on identifying unusual behavior rather than security threats. Sun *et al*. [39] utilizes a support vector machine system to analyze virtual machine usage metrics, operations, monitoring, and updates aimed at anomaly detection. It targets only anomalies related to rolling upgrades, which is one of DevOps operations. Both of these works are limited to AWS virtual machine environments and do not incorporate alternative platforms or types of resources provided by cloud systems. Moreover, they target specific operational scenarios rather than conducting a comprehensive analysis of telemetry logs. Therefore, a systematic analysis of telemetry logs across different platforms and services is still needed to understand correlation between logs and threats.

VII. Conclusion

Cloud telemetry logs, provided by cloud vendors either by default or as optional features, offer a convenient and vendor-integrated mechanism for monitoring and defending cloud instances. Integrated and/or "implemented" by cloud vendors, their content is customized to include information for reliability and troubleshooting purposes. In this paper, we systematically evaluate the capability and cost-effectiveness of cloud telemetry logs as a viable alternative to traditional network-based and/or host-based IDS tools. Our evaluation shows that 45∼48% of 35 realistic attacks with different types of ATT&CK framework are visible by the default logs. By including additional logs, detection coverage improves significantly to 65∼88%, depending on the platform. Furthermore, we present that a lightweight variational autoencoder can detect these attack-induced logs with up to 65% true positive rate, demonstrating the feasibility of effective telemetry-based cloud defense.

Limitations and Future Work. We adopt a controlled experimental setup with a single cloud subscription and one user per platform to ensure reproducibility and isolate the impact of specific attack behaviors on telemetry logs. While this setup does not aim to exhaustively cover the full diversity of real-world cloud deployments, it enables a focused and systematic analysis of the security-relevant signals in cloud telemetry data in a controlled way. This design serves as a foundational step toward establishing the viability of leveraging cloud telemetry for attack detection across heterogeneous platforms, providing a baseline for future extensions to more complex, multi-tenant environments.

We evaluate both default and extended telemetry log configurations as available across major cloud platforms as of 2024. Given the ongoing evolution and refinement of cloud telemetry by service providers, we acknowledge that the detection surface reflected in these logs may improve over time. Our study offers a timely snapshot of current capabilities and establishes a methodological foundation for longitudinal tracking of telemetry log enhancements and their security effectiveness in future work.

Ethical Considerations. The study is exploratory in nature and does not introduce new attack vectors or vulnerabilities. All experimental work was performed in a fully isolated, controlled cloud environment using our own dedicated accounts and instances. The research dataset was exclusively generated within this restricted experimental setup. Due to the controlled nature of the attacks, we did not find cause for extensively reporting attacks undetected by currently available cloud applications (e.g. AWS CloudWatch Log Anomaly). However, to ensure full transparency, we have submitted our findings to AWS.

Acknowledgment

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government.

The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan. This work was supported through contract CR-100043-23-51577 with the U.S. Department of Energy.

APPENDIX

A. Detailed Log Configuration

While the default log configuration is simply that which is available without any changes made to the log collection process on a single user subscription as of 2024, here we state the exact cloud log configurations utilized for the additional log configuration on each platform.

The GCP platform provides additional audit logs from those enabled by default. We enable the "Admin Read", "Data Read", and "Data Write" audit logs for monitoring resource usages at the organization level. Where "Admin Read" allows logging of organization-level read actions such as listing IAM service accounts or roles and viewing resource configurations, metadata, or security settings. We set the `general_logs` flag to `on` when creating our SQL database to allow for general log forwarding to the cloud platform log collection.

On the AWS platform, to collect additional logs, we enable AWS CloudTrail and logging for all basic events provided by AWS. This enables logging for three "Data events". "S3", "Lambda", and "DynamoDB", for all current and future resources created in each of these categories as well as all regions and functions. The logs collected by this CloudTrail are forwarded to an AWS CloudWatch group where they are able to be easily downloaded by the user. We additionally enable the SQL general logs each time a SQL database is created through AWS, which are forwarded to an AWS CloudWatch group to be analyzed with the platform-generated logs.

On the Azure Platform, we enable Azure diagnostic settings at the subscription level to collect the logs in the categories of "Security", "ServiceHealth", "Alert", "Recommendation", "Policy", "Autoscale", and "ResourceHealth" inside of a resource group workspace. This does not give specific resource-level logs such as data read and write or virtual machine access events. We therefore also enable resource-level diagnostic settings, however, these must be set after a resource has been created. We enable these settings for all created resources directly before any payload event takes place to ensure that all control resources have these additional logs enabled.

B. Parameter Sensitivity Studies

We present parameter sensitivity analyses for the two parameters defined in Section IV.C.

The first parameter is the percentage of test cases that must contain attack-related logs for an attack to be considered visible. In this work, we use a threshold greater than 80%, which, given 10 test cases, requires at least 9 to produce attack-related logs. Figure 1 presents results for thresholds of 50% and above. Illustrating that our tests A threshold of 90% indicates that all test cases for a given attack generated attack-derived logs.

The second parameter is the time window ϵ used in Algorithm 1. For our experiments, we use an adaptive ϵ value to account for differences in cloud provider logging speeds. Specifically, ϵ is computed as the average time between control logs multiplied by a scale factor. In this work, we use a scale factor of 50. Figure 2 presents results for varying scale factors and shows that only AWS in the Additional Logs configuration and Azure in both configurations exhibit changes in visibility. The maximum observed difference is 14%, occurring in the Azure Additional configuration.

Figure 1: Test Case Threshold for Attack Visibility Labeling

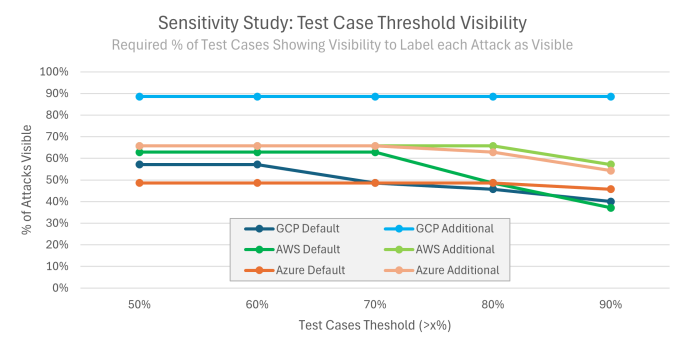
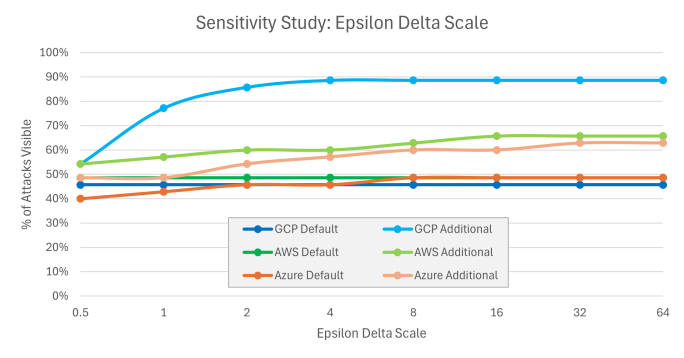


Figure 2: Epsilon Scale for Attack Visibility Labeling



REFERENCES

[1] Bikash Agrawal, Tomasz Wiktorski, and Chunming Rong. Adaptive real-time anomaly detection in cloud infrastructures. *Concurrency and Computation: Practice and Experience*, 29(24):e4193, 2017.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

[3] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special lecture on IE*, 2(1):1–18, 2015.

[4] Behnaz Arzani, Selim Ciraci, Stefan Saroiu, Alec Wolman, Jack Stokes, Geoff Outhred, and Lechao Diwu. {PrivateEye}: Scalable and {Privacy-Preserving} compromise detection in the cloud. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 797–815, 2020.

[5] Microsoft Azure. Azure blob storage pricing, 2025. Accessed: 2025-05-30.

[6] William Blair, Frederico Araujo, Teryl Taylor, and Jiyong Jang. Automated synthesis of effect graph policies for microservice-aware stateful system call specialization. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4554–4572. IEEE, 2024.

[7] Jasmin Bogatinovski, Sasho Nedelkoski, Li Wu, Jorge Cardoso, and Odej Kao. Failure identification from unstable log data using deep learning. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 346–355. IEEE, 2022.

[8] Benjamin Bowman, Craig Laprade, Yuede Ji, and H Howie Huang. Detecting lateral movement in enterprise computer networks with unsupervised graph {AI}. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 257–268, 2020.

[9] Robert A Bridges, Tarrah R Glass-Vanderlan, Michael D Iannacone, Maria S Vincent, and Qian Chen. A survey of intrusion detection systems leveraging host data. *ACM computing surveys (CSUR)*, 52(6):1–35, 2019.

[10] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[11] Yung Ryn Choe, Dalton A. Brucker-Hahn, George Fragkos, Keshav Sreekumar, Bailey Nelson, Alexander Safonov, Ryan Sullivant, Aisha B Rahman, Sean Tsikteris, Md Sadman Siraj, Eirini Eleni Tsiropoulou, Jorge Pineda, Jay Johnson, Brian Tansy, Kyu Hyung Lee, and Junghwan Rhee. Goaltender: Cloud-based defense and response tools for the der ecosystem. In *2024 Resilience Week (RWS)*, pages 1–10, 2024.

[12] Google Cloud. Auditlog api reference, 2025. Accessed: 2025-01-16.

[13] Google Cloud. Logentry api reference, 2025. Accessed: 2025-01-16.

[14] Google Cloud. Structured logging, 2025. Accessed: 2025-01-16.

[15] Pubali Datta, Isaac Polinsky, Muhammad Adil Inam, Adam Bates, and William Enck. {ALASTOR}: Reconstructing the provenance of serverless intrusions. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2443–2460, 2022.

[16] Edge Delta. How many companies use cloud computing in 2024. https://www.edgedelta.com/company/blog/how-many-companies-use-cloud-computing-in-2024.

[17] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1285–1298, 2017.

[18] Henry Hanping Feng, Oleg M Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly detection using call stack information. In *2003 Symposium on Security and Privacy, 2003.*, pages 62–75. IEEE, 2003.

[19] Eric L Goodman, Chase Zimmerman, and Corey Hudson. Packet2vec: Utilizing word2vec for feature extraction in packet data. *arXiv preprint arXiv:2004.14477*, 2020.

[20] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, pages 207–218. IEEE, 2016.

[21] Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[22] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[23] Christopher Kruegel, Darren Mutz, Fredrik Valeur, and Giovanni Vigna. On the detection of anomalous system call arguments. In *Computer Security–ESORICS 2003: 8th European Symposium on Research in Computer Security, Gjøvik, Norway, October 13-15, 2003. Proceedings 8*, pages 326–343. Springer, 2003.

[24] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection with deep learning: How far are we? In *Proceedings of the 44th international conference on software engineering*, pages 1356–1367, 2022.

[25] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1777–1794, 2019.

[26] SC Media. Critical vulnerabilities in 6 aws services disclosed at black hat usa. https://www.scworld.com/news/critical-vulnerabilities-in-6-aws-services-disclosed-at-black-hat-usa, 2024.

[27] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, volume 19, pages 4739–4745, 2019.

[28] Microsoft. Activity log schema, 2025. Accessed: 2025-01-16.

[29] MITRE. Att&ck att&ck. https://attack.mitre.org/.

[30] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. Self-attentive classification-based anomaly detection in unstructured logs. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1196–1201. IEEE, 2020.

[31] Khoi Khac Nguyen, Dinh Thai Hoang, Dusit Niyato, Ping Wang, Diep Nguyen, and Eryk Dutkiewicz. Cyberattack detection in mobile cloud computing: A deep learning approach. In *2018 IEEE wireless communications and networking conference (WCNC)*, pages 1–6. IEEE, 2018.

[32] Harold Ott, Jasmin Bogatinovski, Alexander Acker, Sasho Nedelkoski, and Odej Kao. Robust and transferable anomaly detection in log data using pre-trained language models. In *2021 IEEE/ACM international workshop on cloud intelligence (CloudIntelligence)*, pages 19–24. IEEE, 2021.

[33] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*, 54(2):1–38, 2021.

[34] Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694*, 2019.

[35] SentinelOne. Cloud security issues: 10 critical aspects. https://www.sentinelone.com/cybersecurity-101/cloud-security/cloud-security-issues/.

[36] Amazon Web Services. Cloudtrail event reference, 2025. Accessed: 2025-01-16.

[37] Amazon Web Services. CloudWatch logs anomalies. https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#logsV2:logs-anomalies, 2025.

[38] Amazon Web Services. Guardduty foundational data sources, 2025. Accessed: 2025-05-30.

[39] Daniel Sun, Min Fu, Liming Zhu, Guoqiang Li, and Qinghua Lu. Non-intrusive anomaly detection with streaming performance metrics and logs for devops in public clouds: a case study in aws. *IEEE transactions on Emerging Topics in Computing*, 4(2):278–289, 2016.

[40] Thales. Cloud resources have become biggest targets for cyberattacks, finds thales. https://www.thalesgroup.com/en/worldwide/defence-and-security/press_release/cloud-resources-have-become-biggest-targets.

[41] Abdul Raoof Wani, QP Rana, U Saxena, and Nitin Pandey. Analysis and detection of ddos attacks on cloud computing environment using machine learning techniques. In *2019 Amity International conference on artificial intelligence (AICAI)*, pages 870–875. IEEE, 2019.

[42] Andreas Wespi, Marc Dacier, and Hervé Debar. Intrusion detection using variable-length audit trail patterns. In *Recent Advances in Intrusion Detection: Third International Workshop, RAID 2000 Toulouse, France, October 2–4, 2000 Proceedings 3*, pages 110–129. Springer, 2000.

[43] Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. *ACM SIGARCH Computer Architecture News*, 44(2):489–502, 2016.

[44] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pages 807–817, 2019.

[45] Pengpeng Zhou, Yang Wang, Zhenyu Li, Xin Wang, Gareth Tyson, and Gaogang Xie. Logsayer: Log pattern-driven cloud component anomaly diagnosis with machine learning. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2020.

[46] Bei Zhu, Jing Li, Rongbin Gu, and Liang Wang. An approach to cloud platform log anomaly detection based on natural language processing and lstm. In *Proceedings of the 2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*, pages 1–7, 2020.

[47] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*, 2018.