National College of Ireland

Configuration Manual

MSc Research Project

MSc Cybersecurity

Pranoy Kunhumbiduka Moolakkal
Student ID: x23289597

School of Computing

National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Pranoy Kunhumbiduka Moolakkal

Student ID: x23289597

Programme: MSc Cybersecurity **Year:** 2024-2025

Module: Praticum Part - 2

Lecturer: Niall Heffernan ……

Submission Due Date: 11-08-2025……

Project Title: MSc Research Project

Word Count: ……1078…… **Page Count:** ……6……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Pranoy Kunhumbuduka Moolakkal

Date: 11-08-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Pranoy Kunhumbiduka Moolakkal
X23289597

Title: Enhancing Intrusion Detection and Forensic Readiness Through Cloud Log Redundancy: A Multi-Cloud Security Approach

Objective:
To design and implement an integrated system that ensures real-time redundancy of AWS CloudTrail logs to Microsoft Azure Blob Storage while enabling integrity verification and basic anomaly alerting using hashing and log parsing.

Section 1: AWS Configuration

Create CloudTrail and Enable Logging

1. **Login to AWS Console** and search for "CloudTrail."
2. Navigate to **Trails → Create Trail**.
3. **Trail Name:** cloudtrail-logs
4. **Apply to All Accounts:** Optional (enabled if part of AWS Organizations)
5. **Storage location:**
    - Choose *Create new S3 bucket*
    - Example: aws-cloudtrail-logs-605926690921-xxxxx
6. **Enable Log File Validation**: Yes
7. Click **Next**, choose **Management Events**, and enable **Read and Write** types.
8. Skip Data and Insight events → Review → **Create Trail**

This step ensures that logs from AWS services are captured and stored securely in an S3 bucket in .json.gz format.

Create S3 Bucket Notification Trigger

1. Go to **S3 → Your Bucket** → Select **Properties**
2. Scroll down to **Event Notifications**
3. Create new trigger:
    - Name: send-to-azure
    - Event type: **All object create events**
    - Prefix: (optional) e.g., AWSLogs/
    - Send to: **Lambda Function**
    - Choose the previously created Lambda (will create it below)

Create IAM Role for Lambda

1. Go to **IAM → Roles → Create Role**
2. **Trusted Entity:** Lambda
3. **Permissions:**
    - AmazonS3ReadOnlyAccess
    - CloudWatchLogsFullAccess
4. Create role → Name it lambda-s3-read-role

Section 2: Azure Configuration

Create Azure Storage Account & Blob Container

1. Go to Azure Portal → Create **Storage Account**
    - Name: cloudtraillogsbackup
    - Region: (any)
    - Redundancy: LRS or GRS
2. Once created → Go to **Containers**
    - Create container named: trailbackup
    - Public access level: *Private (no anonymous access)*

2.2 Generate Shared Access Signature (SAS)

1. Navigate to the **Storage Account → Shared Access Signature**
2. Enable:
    - Allowed services: **Blob**
    - Resource types: **Container**, **Object**
    - Permissions: **Read, Write, Create, Add, List**
3. Set start and expiry time (e.g., 1 year validity)
4. Click **Generate SAS token and URL**
5. Copy the **Blob SAS URL** and **SAS token**

Section 3: AWS Lambda Implementation

Create Lambda Function

1. Navigate to **Lambda → Create Function**
    - Runtime: Python 3.12
    - Role: Choose lambda-s3-read-role
    - Name: log_uploader_to_azure
2. Inside **Function Code**, paste the following:

```
import boto3
import requests
import hashlib
import os

def lambda_handler(event, context):
  s3 = boto3.client('s3')
```

```
    # Extract S3 details from event
    bucket = event['Records'][0]['s3']['bucket']['name']
    key    = event['Records'][0]['s3']['object']['key']

    # Fetch the file from S3
    obj = s3.get_object(Bucket=bucket, Key=key)
    data = obj['Body'].read()

    # Compute SHA-256 hash
    sha256_hash = hashlib.sha256(data).hexdigest()

    # Upload to Azure Blob
    sas_url = "https://cloudtraillogsbackup.blob.core.windows.net/trailbackup"
    token = "sv=2024-11-04&ss=b&srt=co&sp=rwactfx&se=2026-07-10T04:02:14Z&st=2025-
07-09T20:02:14Z&spr=https&sig=..."

    # Upload actual log file
    upload_url = f"{sas_url}/{key}?{token}"
    headers = {'x-ms-blob-type': 'BlockBlob'}
    response = requests.put(upload_url, headers=headers, data=data)
    print("Upload status:", response.status_code)

    # Upload the hash
    hash_filename = key + ".hash.txt"
    hash_url = f"{sas_url}/{hash_filename}?{token}"
    requests.put(hash_url, headers=headers, data=sha256_hash.encode())

    return {
        'statusCode': response.status_code,
        'body': ''
    }
```

Every uploaded log file will also generate a .hash.txt file beside it in Azure.
This allows for **integrity verification**.

Section 4: SNS-Based Suspicious Event Alerting

Create SNS Topic & Subscription

1. Go to **SNS → Topics → Create topic**
    - Name: suspicious-events-alerts
2. Create **email subscription**
    - Email: client's email or yours for testing
3. Confirm email by checking inbox

Create Alert Lambda Function

1. Create new Lambda → Name: log_alert_lambda
2. Role: Reuse same role or create new with same policies
3. Paste the following code:

```
import boto3
import gzip
import json

sns = boto3.client('sns')
TOPIC_ARN = "arn:aws:sns:eu-north-1:605926690921:suspicious-events-alerts"

def lambda_handler(event, context):
    s3 = boto3.client('s3')
    bucket = event['Records'][0]['s3']['bucket']['name']
    key    = event['Records'][0]['s3']['object']['key']

    if not key.endswith(".json.gz"):
        return {"statusCode": 200, "body": "Not a log file."}

    obj = s3.get_object(Bucket=bucket, Key=key)
    body = gzip.decompress(obj['Body'].read()).decode('utf-8')
    logs = json.loads(body)

    for record in logs.get('Records', []):
        if record.get("eventName") in ["DeleteTrail", "StopLogging", "CreateUser",
"PutBucketPolicy"]:
            ip = record.get("sourceIPAddress")
            msg = f"Suspicious activity detected:\nEvent: {record['eventName']}\nIP: {ip}\nTime:
{record['eventTime']}"
            sns.publish(TopicArn=TOPIC_ARN, Message=msg)

    return {"statusCode": 200, "body": "Checked."}
```

4. Attach S3 **CreateObject** trigger for .json.gz files in your CloudTrail bucket.

If any suspicious action like DeleteTrail or StopLogging is detected, an alert email will be sent.

Section 5: Automation via Triggers

Upload Trigger (AWS Side)

Already implemented:

- S3 Trigger → Lambda → Azure upload + hash generation

No manual steps needed.

Scheduled SHA / Alert Check (Optional)

If needed, you can create:

- **CloudWatch Events**: Trigger Lambda periodically for additional checks
- **Azure Timer Trigger**: Only if log parsing is shifted to Azure side (currently not used)

You already meet automation goals using **S3 triggers and real-time execution**.

Section 6: Real-Time Log Parsing and Detection via Azure Logic App + Sentinel

This final section implements **real-time log processing and detection** by integrating:

- Azure Logic App (triggered on new blob uploads)
- Azure Function (for secure ingestion)
- Microsoft Sentinel (for log visibility)
- Microsoft Defender (for automated incident detection)

6.1 Create Azure Logic App

1. Go to Azure Portal → Logic Apps → Create Logic App (Consumption plan).
2. Choose Resource Group → Name: `LogParserLogicApp`
3. Location: Same as your storage account
4. Once created, open the Logic App Designer.

6.2 Logic App Workflow Steps

Configure the following steps:

1. **Trigger**: "When a blob is added or modified (properties only)"
   - Storage account: `cloudtraillogsbackup`
   - Container: `trailbackup`
2. **Get blob content using path (V2)**
   - Input: `Path` from previous trigger
3. **Compose**:
   - Expression: `base64ToString(body('Get_blob_content_using_path_(V2)'))`
4. **Parse JSON**:
   - Content: Output of `Compose`
   - Schema: Generate from a sample decoded AWS CloudTrail log (e.g., array of `Records[]`)
5. **For Each**: `item in body('Parse_JSON')?['Records']`
6. Inside the loop:
   - **Azure Function**: `GenerateSignature` (custom)
   - Sends log via HTTP POST to Log Analytics API with secure headers
   - Target: Sentinel Workspace

6.3 Azure Function: GenerateSignature

This function generates required **authorization headers** to ingest logs via the **Log Analytics Data Collector API**.

Ensure it:

- Accepts log record in JSON
- Signs with shared key
- POSTs to Sentinel endpoint with custom log type `AWSCustomLogs_CL`

6.4 Verify in Microsoft Sentinel

1. Go to Microsoft Sentinel → Logs
2. Run KQL query:

```
AWSCustomLogs_CL
| limit 10
```

3. You should see parsed AWS CloudTrail events.

6.5 Create Detection Rule in Microsoft Defender

1. Go to Microsoft Defender → Hunting Rules / Scheduled Analytics Rules
2. Create new rule:
    - **Name**: `AWS Root Login Without MFA`
    - **Query**:

```
AWSCustomLogs_CL
| where userIdentity_type_s == "Root"
| where additionalEventData_MFAUsed_s == "No"
```

- **Alert trigger**: If results found
- **Incident**: Enabled (auto-create)

3. Confirm that when the rule triggers, it appears in **Defender → Incidents**.

Final Outcome

- AWS log generation
- Log fetching & hashing
- Azure upload
- Logic App processing
- Sentinel ingestion
- Defender alerting