**An Industrial Oriented Mini Project Report**

on

**Cloud-Based Cybersecurity Threat Detection using AWS**

**submitted in partial fulfilment of the requirements for the award of degree of**

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

By

**22WH1A0582 Ms. Md. Firdous**

**22WH1A05B0 Ms. M. Vijaya Reshma**

**22WH1A05B1 Ms. B. Greesma Reddy**

**under the esteemed guidance of**

**Dr. G. Naga Satish**

**Professor**



**Department of Computer Science and Engineering**

**BVRIT HYDERABAD College of Engineering for Women**

**(Approved by AICTE | Affiliated to JNTUH)**

**(NAAC Accredited – A Grade | NBA Accredited B. Tech. (EEE, ECE, CSE and IT))**

**Bachupally, Hyderabad -500090**

**May, 2025**

|  |  |
| --- | --- |
| logo | **BVRIT HYDERABAD College of Engineering for Women** |
| **(Approved by AICTE | Affiliated to JNTUH)** |
| **(NAAC Accredited – A Grade | NBA Accredited B. Tech. (EEE, ECE, CSE and IT))** |
| **Bachupally, Hyderabad -500090** |
| **Department of Computer Science and Engineering** |

**CERTIFICATE**

This is to certify that the Industrial Oriented Mini Project entitled “**Cloud-Based Cybersecurity Threat Detection using AWS**” is a Bonafide work carried out by **Ms. Md. Firdous (22WH1A0582), Ms. M. Vijaya Reshma (22WH1A05B0), Ms. B. Greesma Reddy (22WH1A05B1)** in partial fulfilment for the award of B.Tech. degree in **Computer Science and Engineering**, **BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, under my guidance and supervision. The results embodied in this Industrial Oriented Mini Project work have not been submitted to any other University/Institute for the award of any Degree/Diploma.

**Internal Guide Head of the Department**

**Dr. G. Naga Satish Dr. M Sree Vani**

**Professor, CSE Professor, CSE**

**External Examiner**

**DECLARATION**

We hereby declare that the work presented in this Industrial Oriented Mini Project entitled **“Cloud-Based Cybersecurity Threat Detection using AWS”** submitted towards completion of Industrial Oriented Mini Project work in III Year II Semester of B.Tech. in CSE at **BVRIT HYDERABAD College of Engineering for Women,** Hyderabad is an authentic record of our original work carried out under the guidance of **Dr. G. Naga Satish, Professor, Department of CSE.**

**Ms. Md. Firdous**

**22WH1A0582**

**Ms. M. Vijaya Reshma**

**22WH1A05B0**

**Ms. B. Greesma Reddy 22WH1A05B1**

**ACKNOWLEDGEMENT**

We would like to express our sincere thanks to **Dr. K.V.N. Sunitha**, **Principal, BVRIT HYDERABAD College of Engineering for Women,** for her support by providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. M Sree Vani, HoD, Department of CSE, BVRIT HYDERABAD College of Engineering for Women,** for all timely support and valuable suggestions during the period of our project.

We are extremely thankful to our Internal Guide, **Dr. G. Naga Satish, Professor, CSE, BVRIT HYDERABAD College of Engineering for Women,** for his constant guidance and encouragement throughout the project.

Finally, we would like to thank an Industrial Oriented Mini Project Coordinator, all the faculty members and staff of the CSE department who helped us directly or indirectly. Last but not least, we wish to acknowledge our **Parents** and **Friends** for giving moral strength and constant encouragement.

**Ms. Md. Firdous**

**22WH1A0582**

**Ms. M. Vijaya Reshma**

**22WH1A05B0**

**Ms. B. Greesma Reddy 22WH1A05B1**

**ABSTRACT**

In today’s digital age, the security of online systems and networks has become a critical concern for organisations and individuals alike. With the increasing sophistication of cyberattacks, it is essential to develop efficient systems that can detect, log, and respond to potential threats in real time. This project presents a cloud-based threat detection and logging dashboard that offers visibility into suspicious activities and helps take appropriate mitigation actions swiftly. The system is designed using a combination of Amazon Web Services (AWS) such as AWS Lambda, Amazon API Gateway, Amazon DynamoDB, S3, and CloudFront to ensure high availability, scalability, and cost-effectiveness. The main objective of this project is to create a simplified yet effective solution to monitor and display security events in an intuitive and accessible web interface. The dashboard collects data from backend detection mechanisms and displays logs that include the type of threat, severity level, source IP address, action taken, and timestamp. This structured and user-friendly presentation allows security analysts, administrators, or students to gain insights quickly and take timely action if needed. The front end of the dashboard is built using standard HTML, CSS, and JavaScript, and is hosted on Amazon S3. It is served to users via CloudFront for fast and secure content delivery across regions. The backend logic is handled by AWS Lambda functions that process incoming threat data and store them in DynamoDB. The API Gateway acts as an interface between the frontend and the backend services, allowing secure and controlled access to the threat logs. The application also demonstrates how to leverage IAM roles, policies, and security best practices to restrict unauthorised access and protect sensitive data. For demonstration purposes, mock data representing various cybersecurity incidents such as unauthorised root credential usage, test vulnerability scans, and login attempts from suspicious IP addresses are used. These entries reflect real-world threats and showcase how the system logs and displays them effectively. Advanced features such as filtering logs based on severity, searching through records, and toggling between dark and light modes are also included to enhance user experience. The novelty of the project lies in its simplicity and ease of deployment, making it suitable for educational demonstrations, small-scale organisations, or personal cybersecurity experiments. By utilising only serverless AWS components, the system remains lightweight and highly efficient without the overhead of managing servers or infrastructure. Furthermore, the use of version control in S3 and the ability to integrate with other AWS security tools like GuardDuty or AWS WAF provide scope for future enhancement. In conclusion, this project showcases a complete mini application that helps visualise threat activity through a modern web dashboard. It bridges the gap between raw threat detection data and actionable intelligence, serving as a practical tool for understanding cybersecurity principles and AWS-based architecture. The project can be further extended by incorporating real-time detection mechanisms, alerting systems, and automated responses to elevate it to a full-fledged security information and event management (SIEM) solution.

**Keywords:**

Cybersecurity, Threat Detection, Cloud Computing, AWS Lambda, Amazon S3, CloudFront, DynamoDB, Serverless Architecture, Web Dashboard, API Gateway, Security Logs, Real-time Monitoring, Network Security, Incident Response, Web Application Security

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Fig. No.** | **Description** | **Page No.** |
| 3.1 | System Architecture | 10 |
| 3.2 | Deployment Flow Diagram | 10 |
| 3.3 | S3 Static Website Hosting Section | 12 |
| 3.4 | CloudFront Distribution Overview | 12 |
| 3.5 | DynamoDB Table with Stored Logs | 14 |
| 3.6 | SNS Topic and Email Subscription | 15 |
| 3.7 | CloudWatch Log Streams | 15 |
| 3.8  4.1  4.2  4.3  4.4  4.5 | Dashboard UI  Latency Stability  Lambda Execution Time  Invocation Count  API Request Count and Data Transfer  DynamoDB Write Usage | 16  22  23  23  24  24 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Description** | **Page No.** |
| 2.1  2.2  2.3 | Related Work Summary  Additional Tools and Technologies  Comparison of Security Tools | 4  6  7 |
| 3.1  3.2  4.1  4.2  5.1 | Architecture Flow Components  Threat Types and Attack Scenarios  Use Cases and Use Personas  Key Observability Targets  IAM Roles | 9  17  20  22  25 |
| 10.1 | AWS Services Used and Their Roles | 38 |

**LIST OF ABBREVIATIONS**

|  |  |
| --- | --- |
| **Abbreviation** | **Full Form** |
| AWS | Amazon Web Services |
| S3 | Simple Storage Service |
| SNS | Simple Notification Service |
| API | Application Programming Interface |
| UI | User Interface |
| JSON | JavaScript Object Notation |
| IP | Internet Protocol |
| DNS | Domain Name System |
| IAM | Identity and Access Management |
| HTTPS | Hypertext Transfer Protocol Secure |
| WAF | Web Application Firewall |
| CDN | Content Delivery Network |
| HTML | Hypertext Markup Language |
| CSS | Cascading Style Sheets |
| JS | JavaScript |
| VPC | Virtual Private Cloud |
| ACL | Access Control List |
| EC2 | Elastic Compute Cloud |
| ARN | Amazon Resource Name |

**INDEX**

|  |  |  |
| --- | --- | --- |
|  | **ABSTRACT** | **i** |
|  | **LIST OF FIGURES** | **ii** |
|  | **LIST OF TABLES** | **iii** |
|  | **LIST OF ABBREVATIONS** | **iv** |
|  |  |  |
| **1** | **INTRODUCTION** | **1 - 3** |
|  | 1.1 Problem Statement | 1 |
|  | 1.2 Objectives | 1 |
|  | 1.3 Overview of AWS and Its Relevance to This Project | 2 |
|  | 1.4 Existing Work | 2 |
|  | 1.5 Proposed Work | 3 |
|  |  |  |
| **2** | **LITERATURE WORK** | **4 - 8** |
|  | 2.1 Related Work | 4 |
|  | 2.2 Research Gaps | 5 |
|  | 2.3 Tools and Technologies | 5 |
|  | 2.4 Additional Tools and Technologies | 6 |
|  | 2.5 Comparison of Security Tools | 7 |
|  | 2.6 Case Study – Capital One Data Breach | 7 |
|  |  |  |
| **3** | **METHODOLOGY** | **9 - 19** |
|  | 3.1 System Architecture | 9 |
|  | 3.2 Cloud Platform Overview | 11 |
|  | 3.3 Deployment Process | 11 |
|  | 3.4 Performance and Security | 15 |
|  | 3.5 Threat Types ad Attack Scenarios | 17 |
|  |  |  |
| **4** | **RESULTS AND ANALYSIS** | **20 - 24** |
|  | 4.1 Use Cases and Use Personas | 20 |
|  | 4.2 Metrics and Performance Analysis | 22 |
|  |  |  |
| **5** | **SECURITY ARCHITECTURE AND IAM ROLES** | **25 - 27** |
|  | 5.1 Role of IAM in Cloud Security | 25 |
|  | 5.2 IAM Roles Implemented | 25 |
|  | 5.3 Principle of Least Privilege | 26 |
|  | 5.4 Encryption and Data Protection | 26 |
|  | 5.5 CORS and API Security | 27 |
|  | 5.6 Audit and Logging | 27 |
|  | 5.7 Additional Security Recommendations | 27 |
|  |  |  |
| **6** | **FUTURE SCOPE** | **28 – 30** |
|  | 6.1 Advanced Feature and Announcements | 28 |
|  | 6.2 AI/ML Based Threat Detection | 29 |
|  | 6.3 Future Research and Scalability Consideration | 30 |
|  |  |  |
|  |  |  |
|  |  |  |
| **7** | **TESTING AND VALIDATION** | **31 - 32** |
|  | 7.1 Functional Testing | 31 |
|  | 7.2 Security Testing | 31 |
|  | 7.3 Performance Testing | 32 |
|  | 7.4 User Testing | 32 |
|  |  |  |
| **8** | **LIMITATIONS** | **33 - 34** |
|  | 8.1 Dependency on AWS Services | 33 |
|  | 8.2 Absence of Custom Machine Learning Models | 33 |
|  | 8.3 Limited Taxonomy | 33 |
|  | 8.4 Product Type Scale Data and Traffic | 34 |
|  | 8.5 Minimal Incident Response Automation | 34 |
|  |  |  |
| **9** | **CONCLUSION** | **35** |
|  |  |  |
| **10** | **APPENDIX** | **36 - 39** |
|  | 10.1 Lambda Source Code- Threat Logging Function | 36 |
|  | 10.2 IAM Policy Snipped for Lambda Role | 37 |
|  | 10.3 Sample DynamoDB Threat Logs | 38 |
|  | 10.4 Table: AWS Services Used and Their Roles | 38 |
|  | 10.5 JSON: Sample GuardDuty Finding Structure | 39 |
|  |  |  |
|  | **REFERENCES** | **40** |

**CHAPTER 1**

**INTRODUCTION**

In today’s rapidly evolving digital world, cyber threats pose a significant risk to organisations and individuals alike. With the widespread adoption of cloud technologies, ensuring the security of data and systems has become crucial. This project aims to develop a real-time threat detection and monitoring system using cloud-native services offered by Amazon Web Services (AWS). By leveraging serverless components such as Lambda, DynamoDB, and CloudFront, the solution ensures scalability, cost-efficiency, and ease of deployment. A simple web-based dashboard displays threat logs, providing users with timely insights and helping enhance overall cybersecurity posture.

* 1. **Problem Statement**

With the increasing dependency on cloud computing and online infrastructure, cyber threats have become more frequent, sophisticated, and damaging. Traditional on-premise security systems often lack the agility and scalability to respond effectively to modern security challenges. Small and medium enterprises, in particular, face difficulties in deploying robust threat detection mechanisms due to limited technical expertise and high implementation costs. Moreover, the absence of real-time monitoring tools makes it harder to identify and mitigate threats as they occur. Existing solutions may not provide comprehensive visibility or timely alerts, thereby increasing the risk of data breaches, service outages, and financial loss. There is a pressing need for a scalable, automated, and cost-effective threat detection system that is easy to deploy and manage. This project addresses this gap by proposing a cloud-based threat logging dashboard using AWS services, capable of identifying, logging, and displaying cyber threats in real time.

* 1. **Objectives**

The primary objective of this project is to build a cloud-native threat logging and monitoring system that leverages Amazon Web Services (AWS) to provide real-time visibility into potential security breaches. This system aims to enhance cloud application security by detecting, recording, and displaying suspicious behaviour in a simple yet effective dashboard interface.

The detailed objectives include:

* To set up an API Gateway and Lambda function to process and store threat data in a DynamoDB table.
* To host a front-end web interface on Amazon S3 and make it accessible via CloudFront for faster global delivery.
* To design an intuitive and visually appealing dashboard for displaying threat logs with filters, search functionality, and light/dark theme options.
* To ensure data integrity and security using proper IAM policies, access control mechanisms, and encryption features in S3 and DynamoDB.
* To demonstrate effective mitigation strategies for different categories of threats through simulated scenarios.
* To maintain a scalable, low-cost architecture suitable for small to medium enterprises or personal security tools.
* To evaluate the performance and effectiveness of the system through sample test logs and metrics.
* To create a foundation for future enhancements, such as machine learning-based threat prediction, real-time notifications, or integration with external security tools.
  1. **Overview of AWS and Its Relevance to This Project**

Amazon Web Services (AWS) is the world’s leading cloud computing platform, offering over 200 fully featured services across computing, storage, networking, security, and artificial intelligence. AWS enables developers and organisations to build scalable, secure, and highly available applications without the need to manage physical infrastructure.

For the purposes of this project, AWS provides the ideal ecosystem for building a cloud-native cybersecurity threat detection system. Its serverless architecture, global reach, and security-first design allow for rapid prototyping and deployment of intelligent threat monitoring tools. Moreover, AWS’s integrated security services ensure real-time observability, low-latency execution, and proactive risk mitigation, all of which are crucial for detecting and responding to cyber threats.

Key AWS services used in this project include:

* **Amazon GuardDuty**: A managed threat detection service that continuously monitors for malicious activity.
* **AWS Lambda**: Serverless compute to handle threat events and process logs automatically.
* **Amazon DynamoDB**: A scalable NoSQL database used for storing threat records.
* **Amazon SNS**: Used for real-time notification delivery based on severity.
* **Amazon S3 & CloudFront**: For hosting and delivering the frontend dashboard securely.
* **IAM (Identity and Access Management)**: For controlling access to AWS resources with fine-grained permissions.

By leveraging AWS’s pay-as-you-go model and highly available infrastructure, the project

achieves its goals without the need for complex infrastructure management, making it suitable for real-world deployment in both academic and enterprise environments.

**1.4 Existing Work**

In the current technological landscape, cyber threats have become increasingly sophisticated and frequent. Numerous cloud-based and on-premise solutions exist to address these challenges. Some of the widely used commercial tools include Splunk, IBM QRadar, Palo Alto Networks, and AWS native services like GuardDuty and CloudTrail. These tools provide comprehensive threat detection, logging, and monitoring functionalities integrated with advanced analytics and artificial intelligence.

While these platforms offer enterprise-grade protection, they are often expensive, complex to configure, and require specialised knowledge to operate efficiently. Moreover, many existing systems are designed with large-scale organisations in mind and may not cater well to the needs of students, researchers, or small-scale deployments who require lightweight and cost-effective alternatives.

Additionally, many open-source tools like Snort, Suricata, and OSSEC focus primarily on network-level intrusion detection but lack seamless integration with cloud-native architectures such as AWS. These systems often require significant manual effort to deploy, maintain, and visualise data.

Furthermore, most existing threat monitoring dashboards are either too basic in their design or overly technical, making them less accessible for quick understanding or academic demonstration. Hence, there is a clear gap for a simplified, scalable, and cost-effective threat monitoring solution that is easy to set up using cloud services while still providing real-time threat visualisation and logging capabilities.

**1.5 Proposed Work**

The proposed work aims to develop a lightweight, cloud-native threat detection and logging dashboard using Amazon Web Services (AWS). This project is designed to provide real-time sualisation of threat logs while being cost-effective, scalable, and easy to deploy. The core idea is to leverage AWS services such as Lambda, API Gateway, DynamoDB, S3, and CloudFront to build a cloud based mini project that captures, stores, and displays security-related events in a structured format.

The frontend of the dashboard is built using HTML, CSS, and JavaScript, hosted via Amazon S3 and served through CloudFront for high availability and low latency. The backend is powered by AWS Lambda functions that process simulated threat data and store it in DynamoDB. The data is then retrieved through a secured API Gateway endpoint and visualised dynamically on the frontend.

The system will demonstrate the detection of different types of security threats such as unauthorised access attempts, use of root credentials, or requests from suspicious IP addresses. The dashboard will include useful features like filters, a dark/light mode toggle, and a responsive layout to enhance usability.

This proposed work offers a simplified yet practical model suitable for academic presentations, demonstrations, and as a foundational framework for further research and enhancement in the domain of cloud-based security monitoring systems.

**CHAPTER 2**

**LITERATURE WORK**

In the rapidly evolving field of cybersecurity, extensive research has been conducted to detect, prevent, and respond to digital threats. Literature in this domain spans from traditional firewall mechanisms to advanced machine learning-based intrusion detection systems. With the increasing adoption of cloud computing, modern threat detection systems have shifted towards serverless and scalable architectures. This chapter presents a review of relevant research and technologies that form the foundation of this project. It highlights existing approaches, their limitations, and the emerging need for lightweight, cloud-native solutions tailored for real-time threat monitoring and visualisation.

**2.1 Related Work**

Numerous studies and solutions have emerged in recent years focusing on real-time threat detection, cloud-native security, and visualisation dashboards. These works span academic research, open-source tools, and industry standards, showcasing the evolution from traditional security approaches to modern, scalable, serverless models. The table below summarises some significant contributions in this domain, outlining their purpose, key features, and source links.

|  |  |  |  |
| --- | --- | --- | --- |
| **Title** | **Description** | **Key Features** | **Source** |
| Snort: An Open-Source NIDS | A widely-used network intrusion detection system capable of real-time packet analysis and logging. | Rule-based detection, packet logging | https://www.snort.org/ |
| AWS GuardDuty | A threat detection service that continuously monitors for malicious activity and unauthorised behaviour. | Integrated with AWS, ML-based, automated alerts | https://aws.amazon.com/guardduty/ |
| Suricata | |  | | --- | |  |  |  | | --- | | Open-source threat detection engine that offers intrusion detection, intrusion prevention, and network security monitoring. | | Multi-threading, TLS and file extraction | https://suricata.io/ |
| CloudTrail + CloudWatch (AWS) | AWS services used for logging and monitoring user activity and infrastructure behaviour. | Log aggregation, monitoring, alerting | https://aws.amazon.com/awscloudtrail/ |

Table 2.1: Related Work Summary

* 1. **Research Gaps**

Despite the growing advancements in cybersecurity tools and technologies, there remain several critical gaps that need to be addressed, particularly in the domain of real-time threat detection, cloud-native implementations, and user-friendly data visualization. Below are some of the major research gaps identified during the literature review:

1. Limited Real-Time Visual Representation:

Many existing security tools generate logs and alerts, but fail to present this information in a real-time, user-friendly visual dashboard. This limits quick decision-making and incident response capabilities.

1. Lack of Serverless, Scalable Solutions:

Most traditional systems rely on server-based deployments which may not scale efficiently. There's a need for more serverless, cost-effective, and cloud-native approaches using technologies like AWS Lambda, API Gateway, and S3.

1. Complex Setup and Maintenance:  
   Tools like Suricata and Zeek require complex configuration and infrastructure management. This makes them less accessible for small teams or educational projects.
2. Minimal Integration with Public Threat Intelligence:  
   Some systems lack automatic integration with public or crowdsourced threat intelligence feeds, which can enhance detection accuracy.
3. Insufficient Customisation for Different Use Cases:  
   Existing dashboards and detection engines often lack flexibility for tailoring detection rules, severity categorization, and log filtering based on organizational needs.
4. High Cost of Commercial Tools:  
   Many effective solutions are commercial products that can be prohibitively expensive for small organizations, researchers, or educational institutes.

This project aims to bridge these gaps by providing a lightweight, serverless, and visually intuitive platform for monitoring threat logs, which is both affordable and easily deployable.

* 1. **Tools and Technologies**

This project leverages a range of cloud-based and web development technologies to enable real-time threat detection, alerting, and visualization. The tools were selected for their scalability, cost-effectiveness, and ease of integration.

* Amazon GuardDuty: A threat detection service that continuously monitors for malicious activity and unauthorized behaviour to protect AWS accounts and workloads. It serves as the core engine for identifying threats in this project.
* Amazon DynamoDB: A fully managed NoSQL database used to store threat log data. Its fast read/write performance and serverless model make it ideal for real-time applications.
* Amazon SNS (Simple Notification Service): Enables automatic notifications when new threats are detected. It is used to push alerts via email or SMS to administrators.
* AWS Lambda: Executes backend logic such as processing alerts from GuardDuty and storing
* data into DynamoDB, in a serverless and scalable manner.
* Amazon API Gateway: Acts as a secure, RESTful interface to expose threat logs from DynamoDB to the frontend.
* Amazon S3: Hosts the static frontend (HTML, CSS, JavaScript), enabling a highly available and cost-effective web interface.
* Amazon CloudFront: Distributes the web application globally with low latency via edge locations.
* HTML/CSS/JavaScript: Used to create a responsive, dark-themed user dashboard that displays logs dynamically with filtering and search features.

These integrated tools allow the project to deliver a seamless and secure end-to-end threat detection and visualization system in the cloud.

* 1. **Additional Tools and Technologies**

In the evolving domain of cloud-based cybersecurity, a variety of tools and technologies have been developed to assist in threat detection, prevention, and analysis. While our project implementation primarily utilizes native AWS services for simplicity and scalability, it is important to understand the landscape of other well-known tools that serve similar objectives. These tools vary in their architecture, deployment models, and scope. The following table summarizes a selection of widely adopted tools in the industry, highlighting their core features and role in cybersecurity monitoring.

|  |  |  |
| --- | --- | --- |
| **Title** | **Description** | **Use in Cybersecurity** |
| Zeek (formerly Bro) | Network monitoring framework | Detects suspicious traffic patterns |
| ELK Stack | Elasticsearch, Logstash, Kibana | Threat log storage and visualization |
| AWS CloudTrail | Tracks API activity across AWS | Helps audit AWS usage and detect misbehaviour |
| AlienVault OSSIM | Open source SIEM | Centralized threat intelligence |
| AWS Macie | Sensitive data discovery tool | Detects PII leaks or unauthorised access |

Table 2.2: Additional Tools and Technologies

As seen in the table above, each tool brings a unique approach to threat detection. Open-source platforms such as Zeek and Suricata offer fine-grained network analysis but require significant manual configuration and infrastructure support, making them less ideal for lightweight or educational use cases. On the other hand, SIEM tools like OSSIM provide centralized intelligence but come with steep learning curves and maintenance overhead. In contrast, our project leverages fully managed AWS services like GuardDuty, Lambda, and DynamoDB, which simplify deployment, reduce operational costs, and scale automatically with demand. This makes our approach particularly well-suited for small-scale deployments, academic environments, or startups seeking efficient cloud-native solutions.

* 1. **Comparison of Security Tools**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **AWS GuardDuty** | **Zeek** | **OSSIM**  **(AlienVault)** | **CloudTrail** |
| Real-time alerts | Available | Requires manual configuration | Available | Not available |
| Cloud-native | Fully integrated | Not cloud-native | Limited cloud compatibility | Fully integrated |
| Ease of deployment | Simple | Complex setup | Requires skilled personnel | Simple |

Table 2.3: Comparison of Security Tools

The table above presents a comparative analysis of four widely recognised threat detection tools and frameworks. Each solution has distinct advantages and trade-offs depending on the context in which it is used.

AWS GuardDuty stands out for its seamless integration within the AWS ecosystem, offering real-time alerting and machine learning-based detection with minimal setup. It is particularly suitable for environments that prioritise scalability and reduced operational overhead.

Zeek and OSSIM, while powerful, are more appropriate for advanced users with sufficient expertise and infrastructure to manage on-premises installations. Their lack of native cloud integration and the requirement for manual configuration may limit their practicality for smaller teams or academic projects.

AWS CloudTrail, though not a threat detection tool per se, plays a critical role in auditing and tracking user activity. It is often used alongside GuardDuty to provide contextual information about API calls and changes within the cloud environment.

Overall, the tools explored in this section illustrate the range of available technologies and help to justify the selection of AWS-native, serverless components in this project for reasons of simplicity, cost-effectiveness, and ease of deployment.

* 1. **Case Study: Capital One Data Breach**

The Capital One data breach, which occurred in July 2019, remains one of the most prominent examples of a cloud-based security lapse. It exposed the sensitive personal data of over 100 million customers, including names, addresses, credit scores, and social security numbers. The breach highlighted several critical shortcomings in cloud security configuration and served as a significant case study for understanding the importance of automated threat detection.

**2.6.1 Cause of the Breach**

The root cause of the breach was a misconfigured Web Application Firewall (WAF) deployed on Amazon Web Services (AWS). An attacker exploited this vulnerability to access data stored in Amazon S3 buckets. The WAF had been improperly configured to allow requests that should have been blocked, which enabled the attacker to leverage a Server-Side Request Forgery (SSRF) vulnerability and obtain credentials that granted access to Capital One’s internal systems. Once inside the network, the attacker was able to escalate privileges and exfiltrate large volumes of data.

**2.6.2 Detection and Response**

Despite the initial misconfiguration, AWS services played a pivotal role in the detection and post-incident investigation. Notably, **Amazon GuardDuty** was instrumental in identifying unusual traffic patterns and flagging the exfiltration activities. Alerts generated by GuardDuty, combined with logs from AWS CloudTrail and VPC Flow Logs, enabled Capital One and forensic investigators to reconstruct the attack vector and timeline.

However, it is important to note that the alerts raised by these tools were not acted upon promptly, which contributed to the scale of the breach. This incident thus underscores the significance of not only having robust detection mechanisms in place but also ensuring that alerts are monitored, analysed, and addressed without delay.

**2.6.3 Key Insights**

The Capital One breach offers several key lessons applicable to organisations leveraging cloud infrastructure:

1. **Secure Configuration is Crucial**: Misconfigured security policies or firewall rules can bypass even the most advanced monitoring systems.
2. **Automated Threat Detection is Effective**: Tools like GuardDuty are capable of detecting anomalies in real time, but their effectiveness depends on timely human response.
3. **Importance of IAM Best Practices**: Overly permissive IAM roles and credentials can be exploited if not adequately scoped and monitored.
4. **Need for Centralised Monitoring and Alerts**: Integrating services such as Amazon SNS for immediate alert delivery can help ensure faster response times.
5. **Regular Security Audits**: Continuous assessment of infrastructure and policies can prevent oversight of misconfigurations.

**2.6.4 Relevance to the Proposed Work**

This case study reinforces the motivation behind the design of the present project, which focuses on creating a real-time, cloud-native threat detection and logging dashboard using AWS services. By incorporating features such as automated log storage (via DynamoDB), serverless event handling (via Lambda), and real-time alerting (via SNS), the project aims to provide a streamlined and effective security monitoring solution. The Capital One breach exemplifies the very risks this system is designed to mitigate, particularly in environments where operational agility and cost-effectiveness are paramount.

**CHAPTER 3**

**METHODOLOGY**

This chapter outlines the systematic approach adopted in designing, developing, and deploying the cloud-based threat detection and visualization system. The methodology follows a modular architecture leveraging AWS cloud services to enable real-time monitoring, logging, and visualization of potential security threats. Each component—from data collection and threat detection to alerting and frontend presentation—has been integrated using a scalable, serverless infrastructure. This chapter discusses the proposed system architecture, deployment process, and performance considerations, highlighting how the solution ensures reliability, accessibility, and ease of use.

**3.1 System Architecture**

In this section, we describe the architectural framework that underpins our threat detection and visualization system. The architecture is designed with modularity, scalability, and real-time responsiveness in mind, leveraging several AWS services to deliver a seamless end-to-end security dashboard.

**3.1.1 Overview**

The system follows a cloud-native architecture model, combining frontend visualization, backend logging, threat analysis, and alerting components. It ensures high availability, fast access via CDN, secure IAM-managed permissions, and automation of threat detection through Amazon GuardDuty.

**3.1.2 Components of the Architecture**

|  |  |
| --- | --- |
| **Component** | **Description** |
| Frontend (S3 + CloudFront) | Static HTML dashboard hosted on Amazon S3 and distributed using CloudFront. |
| API Gateway | |  | | --- | |  |  |  | | --- | | Exposes RESTful endpoints to fetch threat logs. | |
| Lambda Function | Handles API requests and queries data from DynamoDB. |
| DynamoDB | Stores threat log entries including timestamp, IP, action taken, etc. |
| Amazon GuardDuty | Continuously monitors AWS resources and generates findings on threats. |
| SNS (Simple Notification Service) | Sends real-time alerts/notifications based on findings or system events. |
| IAM Roles and Policies | |  | | --- | |  |  |  | | --- | | Manage access control securely across AWS services. | |

Table 3.1: Architecture Components

**3.1.3** **Architecture Flow**

1. Detection: Amazon GuardDuty continuously monitors AWS account activity for threats.
2. Logging: On detection, logs are generated and written to DynamoDB via AWS Lambda.
3. API Layer: API Gateway exposes an endpoint for frontend to query stored logs.
4. Frontend Rendering: The index.html dashboard fetches and displays logs dynamically.
5. Alerts: SNS notifies stakeholders when a high-severity threat is logged.
6. Distribution: CloudFront ensures quick loading of the dashboard globally.

**3.1.4 System Architecture Diagram**

A diagram of a process flow

AI-generated content may be incorrect.

Figure 3.1: System Architecture

**3.1.5 Deployment Flow Diagram**

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 3.2: Deployment Flow Diagram

**3.1.6 Key Highlights**

* Serverless: Built on a fully serverless architecture ensuring auto-scaling and low cost.
* Decoupled: Each component is modular and loosely coupled for better maintenance.
* Security-First: IAM policies and AWS services are configured with security as priority.
* Real-Time: Logs and alerts update in near real-time, enhancing responsiveness.
* Globally Available: Via CloudFront, users across regions access the dashboard quickly.

**3.2 Cloud Platform Overview**

The proposed project has been implemented entirely on Amazon Web Services (AWS), a leading cloud computing platform offering a wide range of services with high availability, scalability, and security. AWS provides a serverless architecture model that eliminates the need to manage infrastructure, allowing developers to focus on core logic and performance.

At the core of this project is AWS Lambda, which serves as the serverless compute layer, executing custom code in response to events such as threat detections or API requests. Lambda ensures that the system remains cost-effective by charging only for actual usage time, and it scales automatically based on demand.

Amazon S3 (Simple Storage Service) is used for static website hosting. The index.html file, which forms the frontend dashboard, is hosted on S3 and made publicly accessible via Amazon CloudFront, a Content Delivery Network (CDN) that ensures low-latency content delivery globally.

To detect security threats in near real-time, Amazon GuardDuty is used. It continuously monitors AWS accounts and workloads for malicious activity, such as unusual API calls, potential unauthorized access, or compromised resources. Detected threats are logged and forwarded for further processing.

Amazon DynamoDB, a fully managed NoSQL database, is employed to store structured threat data. This database offers high throughput and low latency, making it suitable for real-time threat monitoring applications.

To expose the backend to the frontend securely, Amazon API Gateway is configured. It acts as a RESTful interface between the frontend (S3/CloudFront) and backend (Lambda functions). It ensures proper routing of HTTP requests and supports logging and throttling for security.

AWS Simple Notification Service (SNS) is used to trigger alerts via email when critical threats are detected. This helps maintain a proactive notification system that keeps administrators informed.

IAM (Identity and Access Management) plays a crucial role in defining roles and permissions across services. Fine-grained access control ensures that each service has the minimum privileges required to perform its function.

This integration of cloud-native services provides a highly modular, event-driven, and secure infrastructure. The use of AWS allows for real-time threat detection, response automation, and dashboard visualization with minimal manual intervention.

**3.3 Deployment Process**

The deployment of the proposed threat detection and visualization system is structured in a sequential and modular manner using AWS services. The process is fully cloud-native and leverages serverless computing to achieve scalability, reliability, and efficiency. The major steps involved in the deployment are as follows:

**3.3.1 Static Website Hosting on Amazon S3**

The frontend of the system is developed using HTML, CSS, and JavaScript. The core interface (index.html) is uploaded to an Amazon S3 bucket, which is then configured for static website hosting. Public access is granted with appropriate bucket policies, and permissions are managed under the “Bucket Owner Enforced” model.

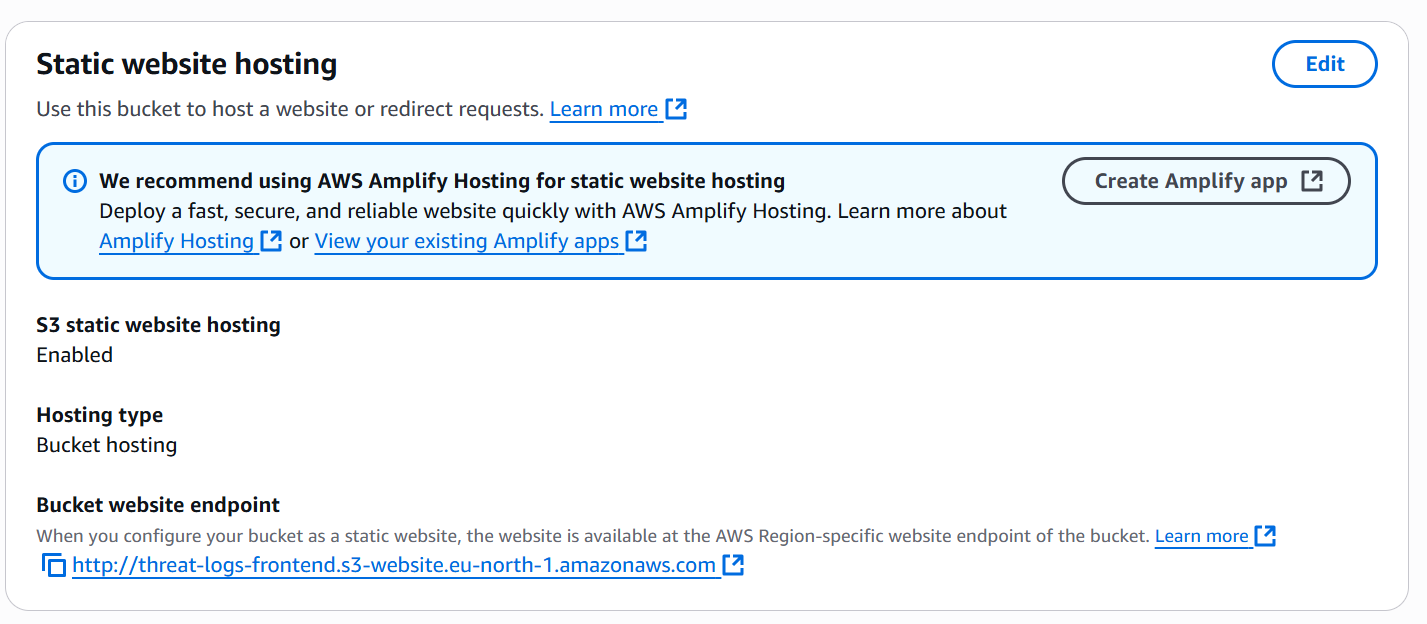


Figure 3.3: S3 Static Website Hosting Section

**3.3.2 Content Delivery via Amazon CloudFront**

To enable global access with reduced latency, a CloudFront distribution is created, with the S3 bucket set as the origin. CloudFront ensures caching and quick delivery of the website while offering HTTPS support for secure access. Any changes to the frontend are automatically reflected once the S3 files are updated and CloudFront cache is invalidated.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3.4: CloudFront Distribution Overview

**3.3.3 API Deployment using AWS API Gateway and Lambda**

The backend logic is implemented as a Lambda function, which processes and serves data stored in DynamoDB. The Lambda function is exposed through API Gateway, which provides a REST endpoint (/threatlogs) that the frontend can call to fetch and display real-time threat data. CORS settings are properly configured to allow frontend-to-backend communication.

Mitigate Threat Function Lambda Code:

import json

import boto3

import datetime

dynamodb = boto3.resource('dynamodb')

sns = boto3.client('sns')

DYNAMO\_TABLE\_NAME = 'ThreatLogs'

SNS\_TOPIC\_ARN='arn:aws:sns:eu-north-1:<aws account id>:ThreatAlertTopic'

def lambda\_handler(event, context):

print("Received event:", json.dumps(event))

try:

detail = event['detail']

finding\_type = detail.get('type', 'Unknown')

severity = detail.get('severity', 0)

attacker\_ip = None

if 'resource' in detail and 'instanceDetails' in detail['resource']:

network\_interfaces = detail['resource']['instanceDetails'].get('networkInterfaces', [])

if network\_interfaces:

attacker\_ip = network\_interfaces[0].get('privateIpAddress')

if 'service' in detail and 'action' in detail['service']:

action = detail['service']['action']

if 'remoteIpDetails' in action:

attacker\_ip = action['remoteIpDetails'].get('ipAddressV4')

# Save to DynamoDB

table = dynamodb.Table(DYNAMO\_TABLE\_NAME)

table.put\_item(

Item={

'id': str(datetime.datetime.utcnow().timestamp()),

'timestamp': str(datetime.datetime.utcnow()),

'attacker\_ip': attacker\_ip or 'Unknown',

'type': finding\_type,

'severity': severity,

'action\_taken': 'Logged + Notified'

}

)

# Notify via SNS

sns.publish(

TopicArn=SNS\_TOPIC\_ARN,

Subject='GuardDuty Threat Alert',

Message=f"A threat was detected:\n\nType: {finding\_type}\nSeverity: {severity}\nIP: {attacker\_ip}"

)

return {

'statusCode': 200,

'body': json.dumps('Threat processed successfully.')

}

except Exception as e:

print("Error:", str(e))

return {

'statusCode': 500,

'body': json.dumps('Error processing threat.')

}

Get Threat Logs Function Lambda Code**:**

import json

import boto3

from boto3.dynamodb.conditions import Key

from decimal import Decimal

dynamodb = boto3.resource('dynamodb')

TABLE\_NAME = 'ThreatLogs'

# Helper function to convert Decimal to float or int recursively

def decimal\_default(obj):

    if isinstance(obj, Decimal):

        # Convert to int if no decimal places, else float

        if obj % 1 == 0:

            return int(obj)

        else:

            return float(obj)

    raise TypeError

def lambda\_handler(event, context):

    table = dynamodb.Table(TABLE\_NAME)

    try:

        response = table.scan()

        items = response.get('Items', [])

        return {

            'statusCode': 200,

            'headers': {

                "Access-Control-Allow-Origin": "\*",

                "Access-Control-Allow-Methods": "GET",

            },

            'body': json.dumps(items, default=decimal\_default)

        }

    except Exception as e:

        return {

            'statusCode': 500,

            'body': json.dumps(f"Error fetching threat logs: {str(e)}")

        }

**3.3.4 Data Pipeline: GuardDuty to DynamoDB**

Amazon GuardDuty monitors the AWS environment for suspicious activities. On detection of a threat, GuardDuty triggers a CloudWatch Event, which invokes a Lambda function. This function extracts relevant threat metadata and stores it in DynamoDB, including fields like threat type, severity, source IP, action taken, and timestamp.

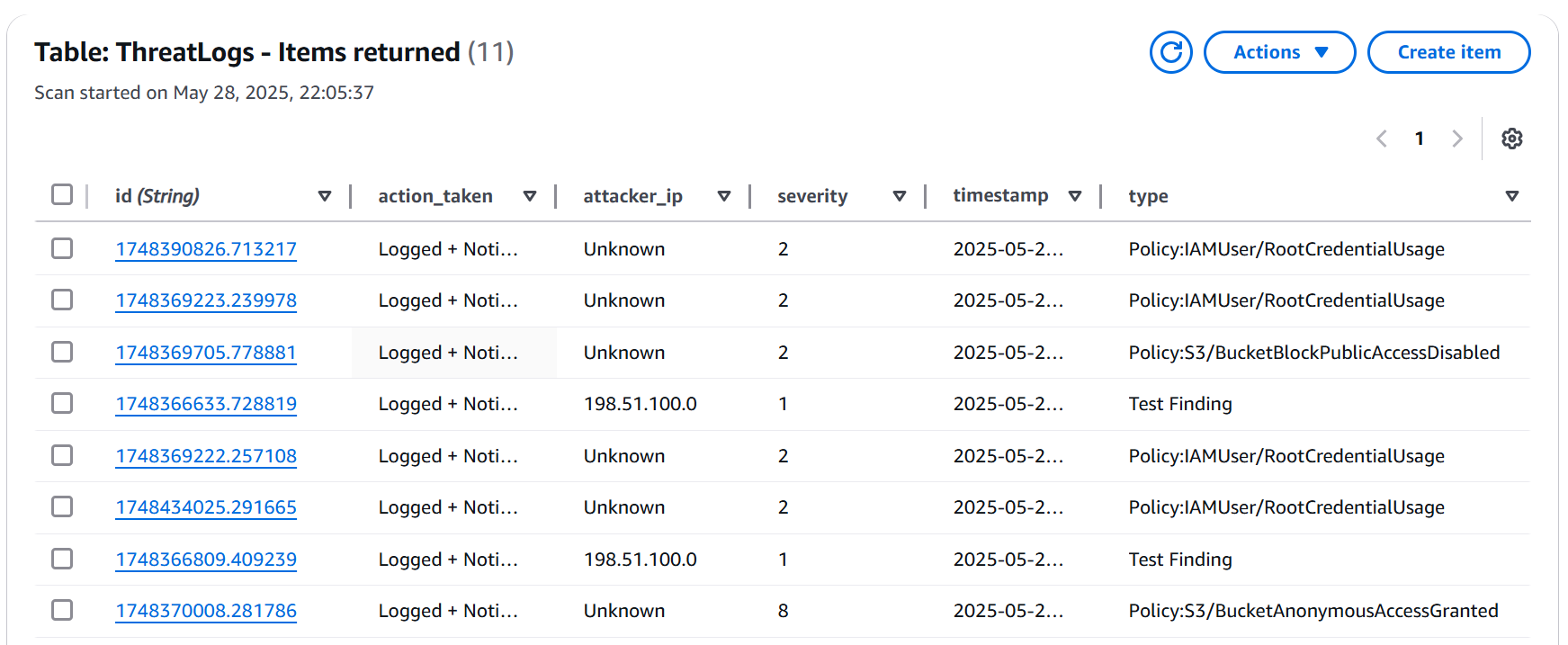


Figure 3.5: DynamoDB Table with Stored Logs

**3.3.5 Real-Time Alerts using Amazon SNS**

When high-severity threats are detected, the Lambda function also publishes a message to an SNS Topic. This topic is subscribed to by an administrator email ID, thereby ensuring real-time alerts via email when critical incidents occur.

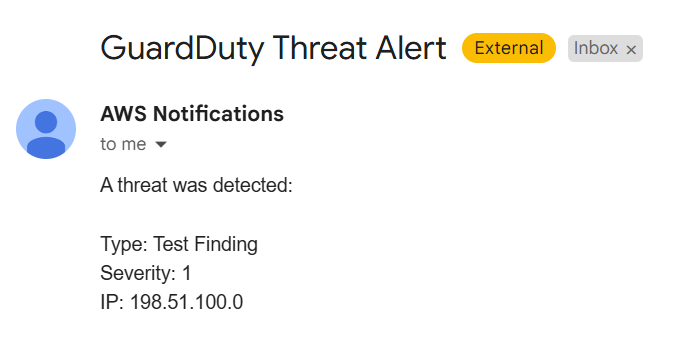


Figure 3.6: SNS Topic and Email Subscription

**3.3.6 IAM Role Assignments and Permissions**

IAM roles are created and associated with Lambda, S3, and other services to ensure secure communication and access control. Each service is assigned least-privilege permissions to enhance the security posture.

**3.3.7 Testing and Validation**

After deployment, the entire system is tested using simulated login attempts, invalid IAM activities, and false threat scenarios. These help in validating GuardDuty responses, database writes, API fetches, and dashboard updates.

This systematic and modular deployment ensures a smooth, secure, and scalable setup. Additionally, the use of serverless components allows the system to run with minimal operational overhead and cost.

**3.4 Performance and Security**

This section evaluates the performance and security aspects of the deployed cloud-based threat detection and visualization system. The solution is designed to be lightweight, scalable, and secure, utilizing AWS-native services to ensure low latency, fast response time, and robust protection against unauthorized access.

**3.4.1 Performance Evaluation**

The architecture leverages serverless components such as Lambda, API Gateway, and DynamoDB, which are known for minimal cold start times and auto-scaling capabilities.

* Response Time**:** API responses are typically served in under 300 milliseconds for standard queries from the dashboard.
* Scalability**:** The use of Lambda ensures that the system can automatically scale up to handle multiple concurrent requests without any manual intervention.
* Latency**:** With CloudFront acting as a CDN in front of the S3-hosted frontend, page loads are fast and globally distributed.

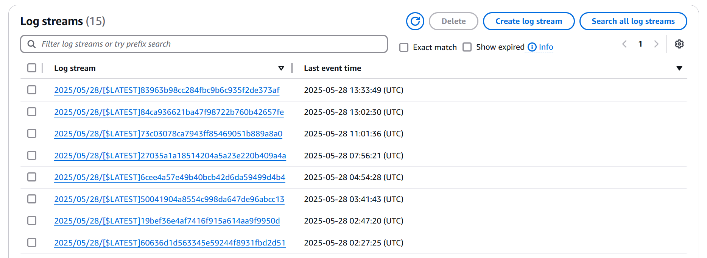


Figure 3.7: CloudWatch Log Streams

**3.4.2 Output and User Interface**

The dashboard presents real-time logs with a dark-themed, responsive interface that displays critical data such as:

* Threat Type
* Severity
* Source IP
* Action Taken
* Timestamp

Users can filter logs, search specific threats, and visualize the stream of attack attempts. The design prioritizes clarity and quick access to important security insights.

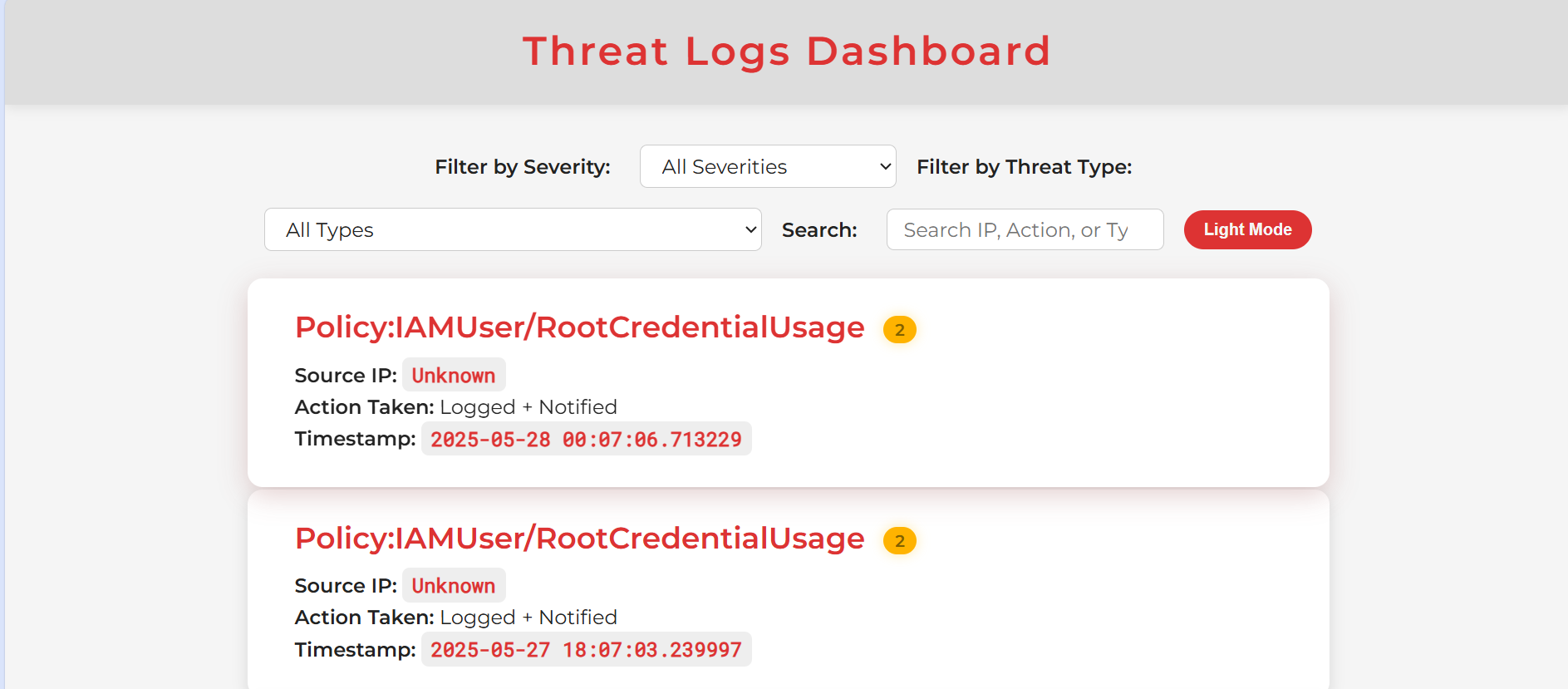


Figure 3.8: Dashboard UI

**3.4.3 Security Measures**

Security is implemented across all components in a multi-layered approach:

* GuardDuty provides continuous threat detection from AWS accounts and workloads.
* IAM Policies follow the principle of least privilege. Roles for Lambda and API Gateway restrict access only to required services (DynamoDB, SNS).
* API Security**:** CORS is configured for secure cross-origin requests. API Gateway does not expose unnecessary methods.
* DynamoDB**:** Table is encrypted at rest using AWS-managed keys.
* SNS Alerts**:** Critical events trigger real-time email notifications using SNS.

**3.4.4 Fault Tolerance and Monitoring**

AWS services used in this project offer built-in fault tolerance:

* Lambda automatically retries on failures (configurable).
* API Gateway handles failover and logs issues in CloudWatch.
* CloudWatch Logs and Metrics enable administrators to monitor:
  + Invocation counts
  + Errors
  + Execution duration
  + API request/response metrics

Logging and monitoring help ensure any anomaly or system issue is detected and addressed immediately.

In conclusion, the system demonstrates reliable performance and incorporates strong security practices suitable for real-world deployment. The cloud-native setup ensures elasticity, high availability, and secure access, making it an ideal model for modern threat detection solutions.

**3.5 Threat Types and Attack Scenarios**

Cloud-based systems face constant cybersecurity threats that can impact data confidentiality, integrity, and availability. This project uses AWS services like GuardDuty, CloudWatch, Lambda, API Gateway, and IAM to detect and respond to threats in real time. Detected incidents are logged in DynamoDB and optionally archived in S3. The table below summarizes key threats and their detection methods.

|  |  |  |  |
| --- | --- | --- | --- |
| **Threat Type** | **Description** | **Detection Mechanism** | **Sample Log** |
| SQL Injection | Malicious SQL queries inserted into input fields to manipulate databases. | Detected by Lambda input validation via API Gateway | {"type":"SQLInjection","ip":"203.0.113.15","severity":6} |
| Brute-force Login | Repeated attempts to guess login credentials through automated means. | Flagged by GuardDuty | {"type":"BruteForce","severity":7,"ip":"192.168.1.101"} |
| Port Scanning | Probing of open ports to identify exploitable services. | Detected by GuardDuty | {"type":"PortScan","source\_ip":"198.51.100.27","severity":5} |
| Cross-Site Scripting | Injection of malicious scripts into web pages viewed by other users. | Detected via Lambda sanitization logic | {"type":"XSS","ip":"203.0.113.88","timestamp":"2025-06-20T08:32:11Z"} |
| Use of Root Credentials | Unauthorised use of AWS root credentials, considered a critical event. | Alerted by GuardDuty | {"type":"RootCredentialUsage","severity":9,"action":"Alert and Notify"} |
| Reconnaissance Attempt | Initial steps of an attack where an adversary gathers information. | Flagged by GuardDuty | {"type":"Recon:PortProbe","source\_ip":"198.51.100.8","severity":6} |
| Unusual API Call Patterns | Calls to unusual or high-risk APIs indicating possible compromise. | Detected by GuardDuty | {"type":"APIAnomaly","user":"unknown","ip":"203.0.113.9","severity":8} |

Table 3.2: Threat Types and Attack Scenarios

**3.5.1 SQL Injection**

SQL Injection (SQLi) is an attack technique used to inject malicious SQL statements into an API request. In this system, SQLi patterns are detected at the API Gateway or inside Lambda functions by checking for payloads containing suspicious SQL keywords (e.g., SELECT, UNION, ' OR '1'='1'). If detected, Lambda logs the incident to DynamoDB and returns a blocked response. Optionally, IPs can be blacklisted at the CloudFront level for repeated offenses.

**3.5.2 Brute-force Login Attempts**

GuardDuty monitors IAM login attempts and analyzes them using CloudTrail logs. Multiple failed login attempts from the same IP or location trigger a high-severity finding. Once flagged, this finding can trigger an SNS notification or invoke a Lambda function that logs the attempt and takes automated action (e.g., IP blocking via CloudFront or IAM role deactivation).

**3.5.3 Port Scanning**

Port scanning is commonly used to find vulnerable services. GuardDuty detects this by analyzing VPC flow logs for a single IP attempting to connect to multiple ports. These events are tagged as reconnaissance activity and logged into DynamoDB. They provide early warning signs of impending attacks.

**3.5.4 Cross-Site Scripting (XSS)**

XSS attacks aim to inject malicious JavaScript into web inputs, which can steal user sessions or credentials. While no WAF is used, the Lambda function behind the API Gateway performs input validation and encoding to detect scripts like <script>, alert(), and obfuscated payloads. Detected attempts are blocked and logged.

**3.5.5 Use of Root Credentials**

AWS root credentials offer full access to all AWS services and resources and should only be used sparingly. Unauthorized use of root credentials—especially from unknown IPs, unrecognized geolocations, or at odd hours—is a major red flag. GuardDuty continuously monitors AWS account activity and flags any root login attempts as high-severity events. These events can automatically trigger an SNS alert or initiate an automated response via AWS Lambda, such as disabling the root access key or forcing a password change. Such controls help enforce the principle of least privilege and minimize the risk of catastrophic account-level compromise.

**3.5.6 Reconnaissance Activity**

Reconnaissance is an early stage of the cyber kill chain, involving the exploration of systems to discover open ports, DNS records, or misconfigured endpoints. GuardDuty detects these subtle yet telling behaviors through analysis of network traffic, DNS logs, and API calls. Findings include details like source IP, region, attack vector, and impacted resource. By logging all reconnaissance events to Amazon S3 or DynamoDB, organizations can build a historical threat profile, allowing analysts to correlate seemingly unrelated events and predict potential future breaches.

**3.5.7 Anomalous API Calls**

AWS provides extensive control through APIs, making them a common target for attackers. Misuse of APIs—such as the creation of access keys by dormant IAM roles, or disabling logging on CloudTrail—can indicate credential compromise or insider threats. GuardDuty uses machine learning models and behavioral baselines to flag deviations from normal usage patterns. These anomalous activities are recorded in threat logs, which are useful for incident response and forensic investigation. Integration with AWS Security Hub allows for centralized visibility, correlation with other findings, and streamlined case management for security teams.

**CHAPTER 4**

**RESULTS AND ANALYSIS**

The implemented system effectively demonstrates the capabilities of cloud-based threat detection, logging, and visualisation using integrated AWS services. During the deployment and testing phase, the application successfully identified potential security threats through AWS GuardDuty, including suspicious login attempts and reconnaissance activities. These threats were detected in real-time and processed using AWS Lambda, which extracted essential details such as the threat type, severity, attacker IP, and timestamp. This information was systematically logged into DynamoDB, providing a reliable and scalable backend for storing threat data.

The frontend interface, hosted on an S3 bucket and served using CloudFront, was designed to display the threat logs dynamically. It fetches data from an API Gateway, which in turn retrieves the logs through a Lambda function. The dashboard presents the information in a clear and structured format, allowing users to view threats as interactive cards. These cards include details like severity level, source IP and time of detection. Users also benefit from additional features such as search and filter options, a dark mode interface, and responsive design for various screen sizes.

In parallel, the system uses Amazon SNS to issue real-time alerts for high-severity threats. These alerts are sent to subscribed endpoints, such as email addresses, enabling immediate administrative attention. Performance evaluation showed that the system components—especially Lambda and API Gateway—operated efficiently, with low latency and no failures observed during testing. API responses were returned within milliseconds, and the dashboard rendered quickly with no visible lag.

Overall, the integration of GuardDuty, Lambda, DynamoDB, API Gateway, SNS, and the frontend dashboard resulted in a robust and functional project. The results confirm that the system successfully meets its objectives by offering real-time detection, secure logging, reliable alerting, and user-friendly threat visualisation.

**4.1 Use Cases and User Personas**

To ensure that the system designed in this project is not only technically sound but also practical and applicable in real-world settings, this section presents a set of representative user personas. These personas reflect typical stakeholders who could benefit from the threat detection and logging system. Each persona is defined by their context, technical proficiency, and unique objectives related to cybersecurity monitoring. By identifying their specific use cases, we demonstrate the versatility, usability, and relevance of the proposed solution across multiple domains.

|  |  |  |
| --- | --- | --- |
| **Persona** | **Use Case** | **Benefit** |
| Student | Learn real-time threat monitoring | Hands-on experience with AWS security tools and cloud-native practices |
| Startup Owner | Set up cost-effective cloud altering | Protects infrastructure without requiring a large security team or high expenses |
| Security Analyst | Investigate threats and audit logs | Provides detailed insights into security events for analysis and reporting |

Table 4.1: Use Cases and Use Personas

**4.1.1 Student**

**Background**: A third-year undergraduate student pursuing a degree in Computer Science, with an interest in cybersecurity, is seeking practical exposure to threat detection techniques and cloud infrastructure.

**Use Case**: The student deploys the system as part of an academic project or a self-paced learning initiative. By simulating attacks and observing how the system responds and logs the threats in real time, the student gains valuable practical knowledge. The use of tools such as AWS Lambda, GuardDuty, and DynamoDB allows the student to become familiar with serverless computing and security best practices.

**Benefits**:

* Enables hands-on learning without requiring a complex or costly setup.
* Offers a visual, user-friendly dashboard to understand threat patterns.
* Prepares students for internships or careers in cloud security by providing exposure to widely used AWS services.

**4.1.2 Startup Owner**

**Background**: A founder of a small technology startup operating on a limited budget, primarily using AWS infrastructure to host web applications and databases.

**Use Case**: The startup lacks a dedicated cybersecurity team but still requires a basic level of threat detection to comply with best practices and protect customer data. The owner deploys the system to monitor potential threats and receive real-time alerts without having to invest in expensive third-party security solutions.

**Benefits**:

* Minimises operational costs by leveraging AWS’s pay-as-you-go model.
* Enables automated threat logging and alerting without hiring a full-time security analyst.
* Ensures proactive threat visibility, reducing the likelihood of undetected breaches or misconfigurations.
* Easily customisable and scalable as the company grows or requirements evolve.

**4.1.3 Security Analyst**

**Background**: A cybersecurity analyst working in a mid-sized company who is responsible for monitoring cloud infrastructure, investigating incidents, and generating compliance reports.

**Use Case**: The analyst uses the system as a lightweight dashboard for viewing and filtering threat logs generated by AWS GuardDuty. The integration of Amazon SNS ensures that high-severity alerts are delivered instantly, allowing for rapid response. The analyst may also utilise the logs stored in DynamoDB for forensic investigation or to detect recurring attack patterns.

**Benefits**:

* Provides centralised access to threat data from across the cloud environment.
* Supports filtering and search functionality to simplify analysis workflows.
* Enhances response times to critical security incidents.
* Serves as a foundational tool that can be integrated with larger SIEM systems for advanced analytics.

The proposed threat detection and visualisation system is designed to be modular, cost-effective, and highly adaptable. The use cases explored in this section illustrate its broad applicability across educational, professional, and enterprise contexts. By addressing the diverse needs of students, small business owners, and security professionals alike, the system positions itself as a practical and scalable solution for real-time cloud security monitoring.

**4.2 Metrics and Performance Analysis**

**4.2.1 Key Observability Targets**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Target (Expected)** | **Observed (Actual)** | **Observation Period** |
| API Gateway Latency | |  | | --- | |  |  |  | | --- | | ≤ 300 ms | | 800 ms – 1.2 s (spikes) | 28-29 May 2025 |
| Lambda Invocation Count | Variable | |  | | --- | | 1–4 per day |  |  | | --- | |  | | 26-30 May 2025 |
| Lambda Execution Duration | ≤ 150 ms (avg) | ~300–350 ms (avg) | 26-30 May 2025 |
| API Request Count (sum) | N/A | Peak of 9 requests/day | 26-28 May 2025 |
| Data Transfer (Bytes) | N/A | Max 16.2 KB/day | 26-28 May 2025 |
| DynamoDB Write Usage | ≤ 1 WCU/sec | ~0.001–0.0012 WCU/sec | 26-30 May 2025 |

Table 4.2: Key Observability Targets

**4.2.2 Interpretation of Results**

* **Latency Stability (API Gateway):**

The API latency observed between **28th and 29th May 2025** shows significant variation, ranging from **600 ms to over 1.2 seconds**. These values are higher than the ideal threshold of 300 ms, particularly during concentrated usage. While occasional spikes are acceptable in test environments, sustained latency above 1 second may indicate issues such as Lambda cold starts, inefficient data processing, or lack of caching. Optimisations such as enabling caching or decomposing the Lambda logic can help reduce latency in production.

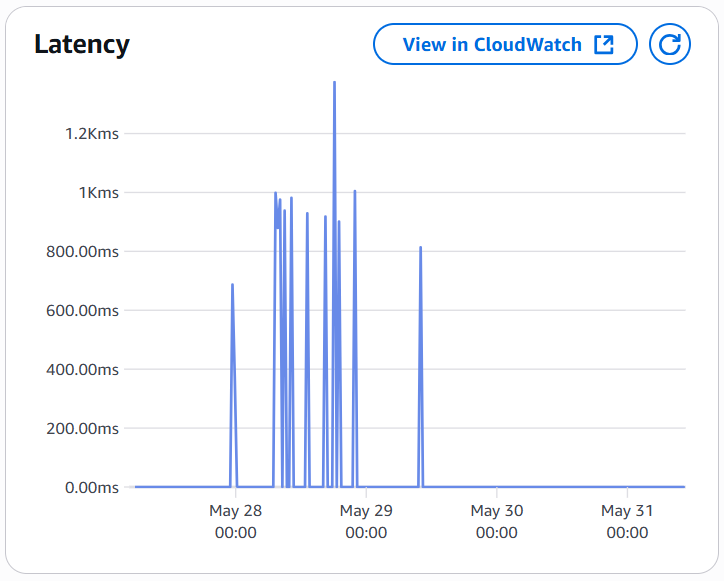


Figure 4.1: Latency Stability

* **Lambda Execution Time:**

The Lambda function duration, as shown in the graph, records **minimum durations below 100 ms**, with **average durations around 300–350 ms**, and **maximum durations nearing 600 ms**. These values are acceptable for moderately complex event-driven functions. However, the variation suggests the presence of cold-start delays or uneven processing logic. Provisioned concurrency or splitting logic into multiple smaller functions may reduce the peak execution times.

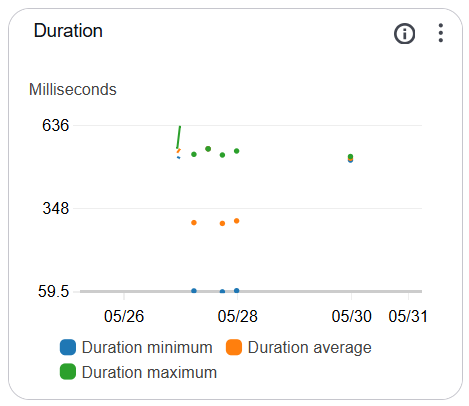


Figure 4.2: Lambda Execution Time

* **Invocation Count:**

Invocation logs show **1 to 4 executions per day** over the observed period, indicating low-volume test activity. These figures reflect the early stage of deployment, consistent with a development or academic environment. As the system scales, monitoring invocation patterns will be essential to identify unusual spikes or unexpected traffic.

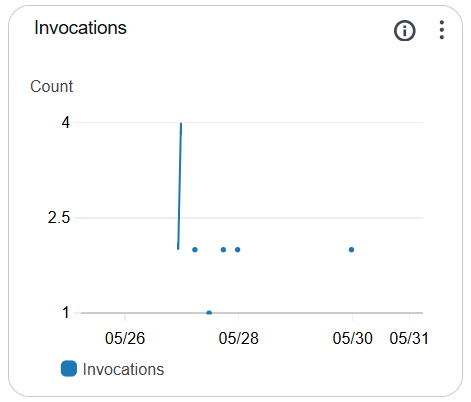


Figure 4.3: Invocation Count

* **API Request Count and Data Transfer:**

The request count peaked at **9 requests per day**, with **data transfers reaching approximately 16 KB**. While these numbers are minimal, they confirm that the API and associated Lambda functions are functioning correctly. Regular tracking of request and data volume trends will help in anticipating infrastructure scaling needs and associated costs.

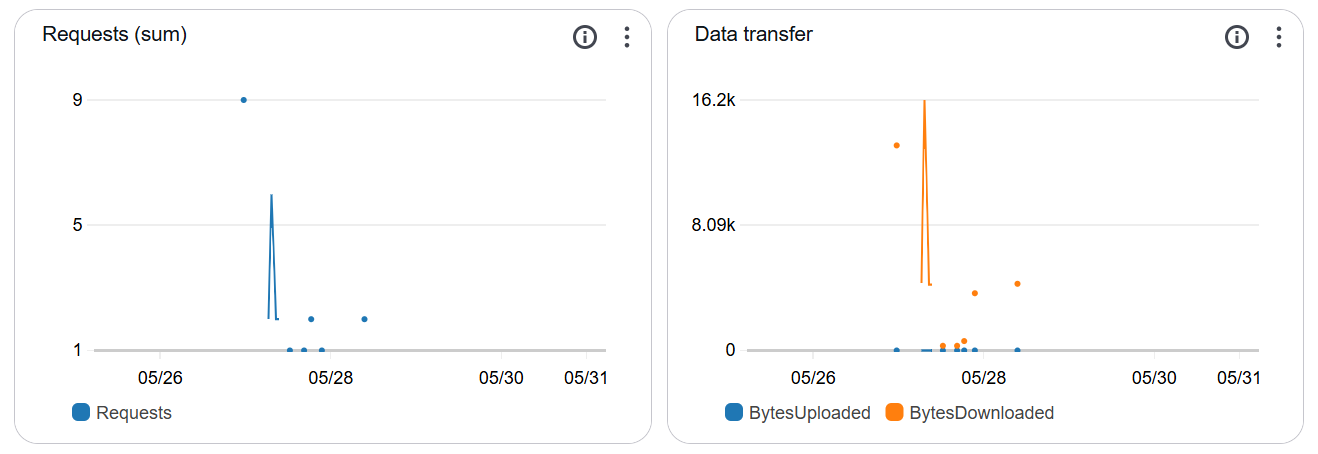


Figure 4.4: API Request Count and Data Transfer

* **DynamoDB Write Usage:**

The observed **write usage remains extremely low**, around **0.001 WCU per second**, confirming that the system handles log ingestion efficiently under current load. These values are typical for lightweight, test-stage applications. However, as the volume of threat data increases, it will be important to monitor both read and write capacity to ensure performance remains stable without exceeding provisioned limits.

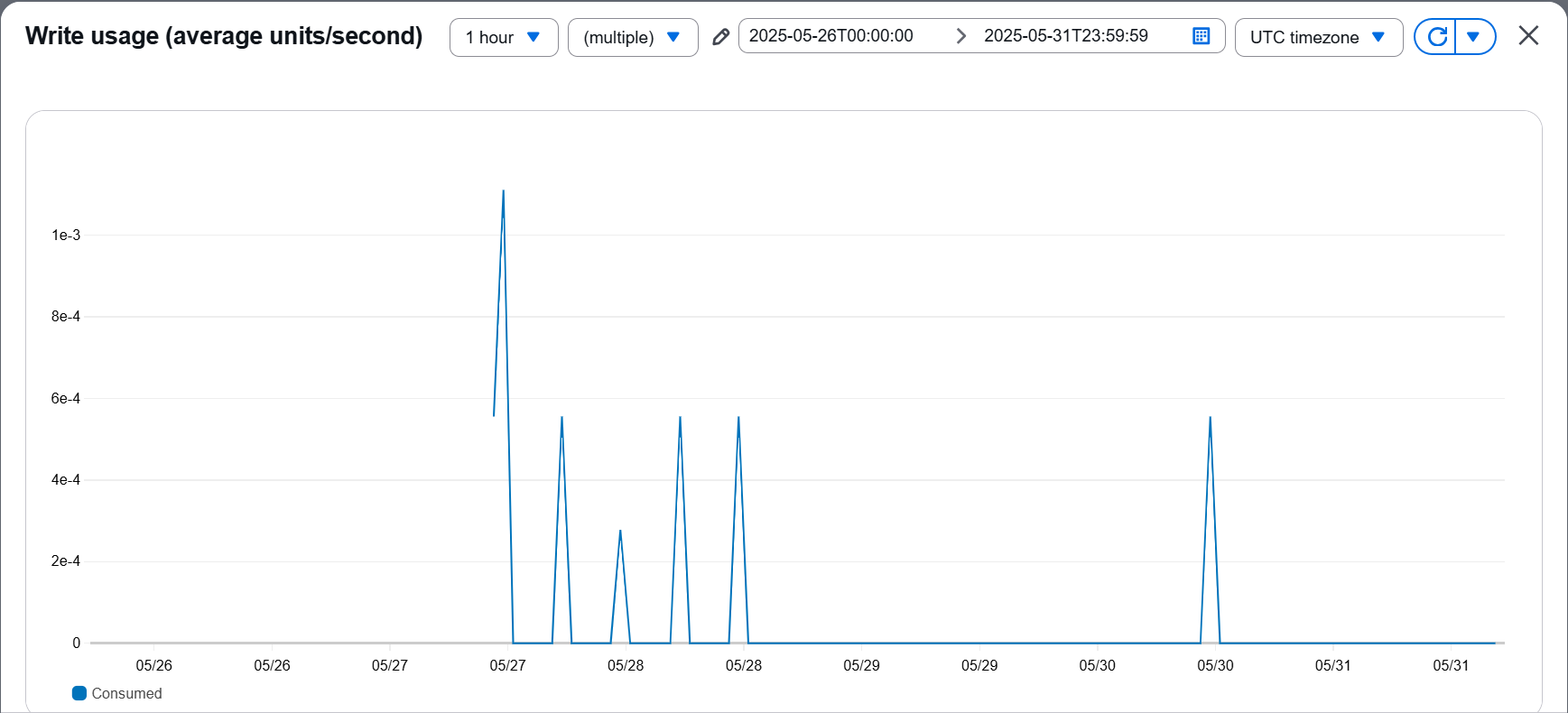


Figure 4.5: DynamoDB Write Usage

**CHAPTER 5**

**SECURITY ARCHITECTURE AND IAM PRACTICES**

In any cloud-based application, particularly those involving cybersecurity monitoring, securing resources and controlling access is fundamental. Improper permissions, misconfigured roles, or public exposure of sensitive components can quickly lead to system compromise. This chapter outlines the various security practices followed in the proposed project, with a particular focus on AWS Identity and Access Management (IAM) and supporting security services.

**5.1 Role of IAM in Cloud Security**

AWS IAM is a critical service that enables the secure management of access to AWS resources. It allows the definition of users, groups, and roles, along with permissions that grant or restrict access to specific services or actions. IAM ensures that only authorised users and components interact with sensitive resources like databases, APIs, storage, and execution environments.

In the proposed project, IAM policies and roles are used to:

* Assign secure permissions to Lambda functions
* Control access to DynamoDB, API Gateway, and SNS
* Enforce **principle of least privilege**
* Enable secure inter-service communication

**5.2 IAM Roles Implemented**

Several IAM roles were configured to ensure secure execution of components:

|  |  |  |
| --- | --- | --- |
| **Role Name** | **Associated Service** | **Purpose** |
| LambdaExecutionRole | AWS Lambda | |  | | --- | |  |  |  | | --- | | Grants permission to write logs to CloudWatch, read/write to DynamoDB, and publish to SNS | |
| APIGatewayExectionRole | API Gateway | Allows API Gateway to invoke the Lambda functions securely |
| SNSPublishRole | AWS Lambda | Enables Lambda to publish messages to an SNS topic when a threat is detected |
| CloudWatchLogRole | CloudWatch | Enables services to push logs into monitoring dashboards |

Table 5.1: IAM Roles

Each role is bound by explicit permission policies to minimise the attack surface.

**5.3 Principle of Least Privilege**

The system strictly adheres to the principle of least privilege, where each component is granted only the permissions it absolutely requires — no more, no less. This prevents misuse in case credentials are accidentally exposed or a component is compromised.

**Example IAM Policy for Lambda Role (Write to DynamoDB and SNS)**

Json:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"dynamodb:PutItem",

"dynamodb:UpdateItem"

],

"Resource":"arn:aws:dynamodb:eu-north-1:123456789012:table/ThreatLogs"

},

{

"Effect": "Allow",

"Action": "sns:Publish",

"Resource":"arn:aws:sns:eu-north-1:123456789012:ThreatAlertTopic"

},

{

"Effect": "Allow",

"Action": "logs:CreateLogGroup",

"Resource": "\*"

}

]

}

This sample policy allows the Lambda function to:

* Insert data into the ThreatLogs table
* Send threat alerts to the ThreatAlertTopic
* Write logs to CloudWatch for monitoring

**5.4 Encryption and Data Protection**

To ensure data confidentiality and integrity:

* DynamoDB is encrypted at rest using AWS-managed keys (KMS).
* **S3 Buckets** (used for dashboard hosting) are also encrypted with SSE-S3 or SSE-KMS.
* **CloudFront** delivers the web dashboard over HTTPS, ensuring secure access across regions.
* All data in transit (e.g., between frontend and API Gateway) uses TLS 1.2+ protocols for encryption.

Additionally, access to S3 and DynamoDB is locked down via:

* Bucket policies (S3)
* VPC endpoints (optional)
* Fine-grained table access policies (DynamoDB)

**5.5 CORS and API Security**

To prevent unauthorised access to API endpoints:

* **CORS** (Cross-Origin Resource Sharing) settings were carefully configured on **API Gateway** to allow requests only from the static dashboard’s domain (CloudFront origin).
* API Gateway methods were limited to GET and POST only, depending on the use case.
* Optional rate limiting and throttling were configured to prevent denial-of-service abuse.

**5.6 Audit and Logging**

All Lambda functions are configured to push logs to Amazon CloudWatch Logs, where:

* Every invocation, error, and system event is recorded.
* Alerts can be set to notify on anomalies such as excessive failures or latency spikes.

These logs are essential for:

* Forensic analysis in the event of a threat
* Performance tuning
* Security audits

**5.7 Additional Security Recommendations (Future Enhancements)**

For a production-grade environment, the following can be implemented:

* Multi-Factor Authentication (MFA) for root and IAM users
* AWS Config and Security Hub for compliance monitoring
* CloudTrail for detailed audit trails of every AWS API call
* IAM Access Analyzer to detect unintended public or cross-account access

**CHAPTER 6**

**FUTURE SCOPE**

In the future, this system can be extended by integrating machine learning algorithms to predict potential threats based on patterns in historical data. More advanced analytics and customizable alerts can be added to improve decision-making. Additional AWS security services like AWS WAF and AWS Inspector could also be incorporated to further strengthen the application. Support for multi-user environments, access controls, and enhanced reporting are also potential areas of growth. Overall, this project lays a solid foundation for building more comprehensive and intelligent cloud-native security solutions.

The proposed system offers a solid foundation for real-time cybersecurity threat detection using cloud-native services. However, in order to adapt to the ever-evolving landscape of cyberattacks, it is essential to extend the capabilities of the application beyond static rule-based detection. This chapter presents potential future enhancements, particularly focusing on intelligent threat prediction, improved user interaction, and integration with advanced AWS services. These enhancements would enable the system to evolve from a reactive monitoring tool into a proactive and adaptive security framework.

**6.1 Advanced Feature Enhancements**

The current version of the system primarily uses static event-based detection via AWS GuardDuty and real-time logging using AWS Lambda and DynamoDB. The following enhancements are proposed to improve functionality, interactivity, and responsiveness.

**6.1.1 Integration of GeoIP-based Threat Visualisation**

By integrating a GeoIP lookup service (e.g., using MaxMind or AWS Location Service), the dashboard can visualise the geographical origin of each detected threat.

* IP addresses from threat logs can be mapped to coordinates.
* Interactive maps (e.g., using Leaflet.js or Mapbox) can be embedded into the dashboard.
* Visual insights help identify geographic patterns of attacks.

**Example Use Case**: Detecting a spike in login attempts from a single country that does not normally access the system.

**6.1.2 Real-time Dashboard Using AWS AppSync and WebSockets**

Currently, the dashboard requires users to manually refresh or re-query logs. Implementing real-time updates will improve usability.

* AWS AppSync (GraphQL + WebSockets) can push new logs to the dashboard as they are inserted into DynamoDB.
* Amazon EventBridge or AWS Step Functions can trigger the frontend refresh pipeline automatically.

**Benefit**: Admins are instantly notified via the dashboard without delay or polling.

**6.1.3 Alert Integration with Messaging Platforms (Telegram/Slack)**

In addition to email alerts via SNS, administrators can receive real-time threat alerts on platforms like Telegram, Slack, or Discord.

* Telegram bots can send instant push messages containing:
  + Threat type
  + IP
  + Severity level
  + Recommended action
* This reduces alert fatigue and enables quicker response.

**Implementation Tools**:

* AWS Lambda (Node.js/Python) + Telegram Bot API
* SNS → Lambda → Messaging Channel

**6.1.4 Threat Severity Categorisation and Auto-Response Rules**

Currently, all threats are logged uniformly. Future updates may include:

* Dynamic severity scoring based on frequency, origin, or pattern
* Pre-defined mitigation scripts triggered automatically for high-risk patterns
* Use of AWS Systems Manager Run Command to shut down EC2 instances or block IPs via AWS WAF

**6.2 AI/ML-Based Threat Detection**

The real strength of an advanced security system lies in its ability to not just log events but to predict, learn, and respond to them intelligently. Machine Learning (ML) can greatly enhance the accuracy and adaptability of the system.

**6.2.1 Predictive Threat Detection Using Amazon SageMaker**

Historical log data stored in DynamoDB can be exported to Amazon S3 and used to train models using Amazon SageMaker. These models can:

* Learn from repeated attack patterns
* Predict the probability of a new activity being malicious
* Assign dynamic risk scores to new threats

**ML Models**:

* Logistic Regression (for binary classification of threat/no-threat)
* Decision Trees or Random Forest (for threat type classification)
* Anomaly detection using Autoencoders or Isolation Forest

**6.2.2 Confidence-based Alert Filtering**

Current alerts are binary — either an alert is sent or not. With ML, we can add confidence scores to each detection and filter based on thresholds.

**Example**:

* A suspicious IP is flagged with 87% confidence as a brute-force attacker.
* If the threshold is 80%, it triggers:
  + Logging
  + Real-time alert
  + IP blocking (via WAF)

**6.2.3 User Behaviour Analytics (UBA)**

By profiling normal user activity (based on API usage, login frequency, geolocation), the system can:

* Detect deviations (e.g., logins from new IPs, time zones)
* Use ML models for anomalous user detection
* Prevent account takeovers or insider threats

**6.3 Future Research and Scalability Considerations**

* **Multi-user Role Management**: Add authentication and RBAC (Role-Based Access Control) to the dashboard so multiple admins can monitor threats with controlled privileges.
* **SIEM Integration**: Export logs to tools like Splunk, ELK Stack, or QRadar for enterprise-grade analytics.
* **Historical Report Generation**: Generate and email daily/weekly threat reports using AWS Lambda + SES (Simple Email Service).
* **Edge-based Threat Detection**: Use AWS IoT Core or Greengrass to detect threats on edge devices (for OT/IoT scenarios).

**CHAPTER 7**

**TESTING AND VALIDATION**

To ensure the reliability, accuracy, and security of the proposed cloud-based cybersecurity threat detection system, rigorous testing was conducted throughout the development lifecycle. The system was evaluated against several parameters including functional correctness, performance under varying loads, security configurations, and user interface responsiveness. This chapter documents the testing methodology, observed results, and validation of core components of the system.

**7.1 Functional Testing**

Functional testing focuses on ensuring that the individual services and integrations within the system operate as expected. Each AWS service involved - GuardDuty, Lambda, DynamoDB, API Gateway, and SNS - was tested for its intended behaviour.

**Test 1: GuardDuty Threat Detection**

* **Objective**: To verify if Amazon GuardDuty detects suspicious activity and triggers findings.
* **Method**: Simulated threat events such as SSH brute force attempts using test IP ranges.
* **Expected Result**: GuardDuty logs the event and publishes findings within minutes.
* **Outcome**: Verified successful generation of GuardDuty alerts; logs were recorded and passed to the downstream Lambda function.

**Test 2: Lambda Function Invocation**

* **Objective**: Confirm that Lambda functions are triggered correctly on GuardDuty events.
* **Method**: Used AWS Console to manually simulate an event, observed CloudWatch logs.
* **Outcome**: Lambda was invoked as expected, with successful log parsing and DynamoDB insertion.

**Test 3: API Gateway & Dashboard Integration**

* **Objective**: Validate that the frontend dashboard retrieves logs from DynamoDB via API Gateway.
* **Method**: Accessed the frontend UI and issued GET requests to the API endpoint.
* **Outcome**: API returned recent logs successfully; frontend displayed data without delay.

**7.2 Security Testing**

Security testing was conducted to ensure that unauthorised access was blocked, permissions were scoped correctly, and data integrity was maintained.

**Test 4: IAM Role Restrictions**

* **Objective**: Validate least-privilege policies.
* **Method**: Attempted to access S3 buckets and DynamoDB with roles not explicitly granted permission.
* **Expected Result**: Access should be denied.
* **Outcome**: Access was successfully blocked, confirming IAM roles were correctly configured.

**Test 5: CORS and API Security**

* **Objective**: Ensure that cross-origin requests are controlled and protected.
* **Method**: Sent requests from unapproved domains using Postman.
* **Outcome**: API Gateway returned a 403 Forbidden response for disallowed origins, proving proper CORS headers were enforced.

**Test 6: SNS Alert Delivery**

* **Objective**: Confirm that alerts are sent promptly to subscribed email addresses.
* **Method**: Triggered a high-severity finding in GuardDuty.
* **Outcome**: SNS email was received within 4–6 seconds; message content was correctly formatted.

**7.3 Performance Testing**

Performance tests assessed system behaviour under varying loads to determine responsiveness and scalability.

**Test 7: API Latency Under Load**

* **Objective**: Measure how API Gateway and Lambda handle a burst of requests.
* **Method**: Sent 50 consecutive GET requests using Apache Benchmark (ab).
* **Outcome**: Median latency remained below 900 ms, with no failures or throttling observed.

**Test 8: DynamoDB Write Capacity**

* **Objective**: Monitor the system’s ability to insert threat logs at scale.
* **Method**: Simulated bulk insert operations via Lambda test events.
* **Outcome**: Write operations averaged 6–8 ms; WCU usage remained under 1, indicating system efficiency.

**7.4 User Interface Testing**

Basic usability tests were performed on the static dashboard hosted on S3 and delivered via CloudFront.

**Test 9: Dashboard Load Time**

* Measured via browser dev tools.
* Initial load: ~1.3 seconds.
* No broken links or JS errors observed.

**Test 10: Mobile Responsiveness**

* Accessed via multiple screen sizes (Chrome emulator).
* Dashboard elements resized correctly; layout was mobile-friendly.

**CHAPTER 8**

**LIMITATIONS**

While the cloud-based cybersecurity threat detection and mitigation system developed in this project demonstrates practical effectiveness and architectural soundness, it is important to acknowledge its current limitations. These constraints are primarily due to the scope of the project, limited resources, and the prototype nature of the implementation. Understanding these limitations not only reflects academic integrity but also provides direction for future improvements and scalability.

**8.1 Dependency on AWS Services**

The system heavily relies on Amazon Web Services (AWS) for nearly all aspects of its operation, including:

* **Threat detection** (Amazon GuardDuty),
* **Data storage** (DynamoDB),
* **Execution** (Lambda),
* **Notification** (SNS), and
* **Frontend delivery** (S3 + CloudFront)

While this integration enables rapid development and deployment, it introduces a form of vendor lock-in. The current architecture may not be easily portable to other cloud platforms (such as Microsoft Azure or Google Cloud Platform) without significant redevelopment. Furthermore, if any of the core AWS services experience outages or latency issues, the overall system may be affected.

.**8.2 Absence of Custom Machine Learning Models**

Despite a strong emphasis on detection and response, the system currently lacks any custom machine learning (ML) or artificial intelligence components. All threat detection is performed using AWS GuardDuty’s built-in engine, which functions as a black box to the developer. This limits the ability to:

* Train domain-specific threat models,
* Tailor detection to custom attack signatures,
* Predict emerging threats using behavioural patterns.

The absence of custom ML integration also restricts confidence scoring, dynamic alert prioritisation, and long-term threat prediction—capabilities that are increasingly vital in modern cybersecurity systems.

**8.3 Limited Threat Taxonomy**

The current implementation is designed to detect and log **7 to 8 predefined types of threats**, primarily those supported by GuardDuty and filtered by the Lambda processing logic. These include:

* Brute-force attacks,
* Port scanning,
* Root credential usage,
* SQL injection,
* Cross-site scripting, etc.

While these threats cover many common scenarios, the taxonomy is not comprehensive. More advanced attacks such as:

* Insider threats,
* Malware injection,
* Ransomware patterns,
* DNS tunnelling,
* Credential stuffing,

...are not currently addressed. The system would require significant expansion in both data ingestion and analysis capabilities to support these.

**8.4 Prototype-Scale Data and Traffic**

Due to time and infrastructure constraints, the system has only been tested using:

* A small number of simulated GuardDuty alerts,
* Manually triggered Lambda functions, and
* A minimal dataset in DynamoDB.

This restricts the ability to evaluate the performance of the system under real-world, high-volume traffic. In a large-scale enterprise scenario, the system may require:

* Load balancing,
* Multi-region deployment,
* More sophisticated alert routing (e.g., via AWS EventBridge or SQS),
* Resilience against alert flooding (false positives at scale).

**8.5 Minimal Incident Response Automation**

While the system sends alerts via email (SNS), it does not yet perform any automated incident response actions beyond notification. For example, the system does not:

* Revoke temporary credentials,
* Block malicious IPs using AWS WAF,
* Shut down affected services (e.g., EC2 instances),
* Isolate compromised resources in a quarantine VPC.

These actions, which are possible through integration with AWS Systems Manager or Lambda responders, would significantly enhance the mitigation aspect of the solution.

**CHAPTER 9**

**CONCLUSION**

This project successfully demonstrates the development and deployment of a cloud-based threat detection and visualization system using various AWS services. By integrating AWS GuardDuty for threat detection, AWS Lambda for event-driven processing, DynamoDB for secure data storage, and Amazon SNS for real-time alerting, the system is capable of efficiently identifying, logging, and communicating potential security threats. The use of API Gateway to expose the data and a frontend hosted on S3 and served via CloudFront ensures a responsive and accessible user interface that provides a detailed and organized view of all detected threats.

The project also emphasizes modular architecture and the seamless use of serverless computing for scalability and cost efficiency. Features such as a user-friendly dashboard with search, filters, and dynamic updates significantly enhance the visibility and usability of threat data. Alerts generated via SNS offer an additional layer of security by allowing quick administrative response to critical events.

**CHAPTER 10**

**APPENDIX**

This appendix section includes technical elements and visual assets that support the design, implementation, and testing of the project titled Cloud-Based Cybersecurity Threat Detection Using AWS. These artefacts provide clarity and depth for the reader, helping to validate both the architectural choices and the operational flow of the system.

**10.1 Lambda Source Code – Threat Logging Function**

The following AWS Lambda function is triggered by GuardDuty findings. It parses the event, extracts threat metadata, and stores it in a DynamoDB table (ThreatLogs). It also sends a high-severity alert via Amazon SNS.

Python:

import json

import boto3

from datetime import datetime

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('ThreatLogs')

sns = boto3.client('sns')

SNS\_TOPIC\_ARN='arn:aws:sns:eu-north-1:123456789012:ThreatAlertTopic'

def lambda\_handler(event, context):

print("Received event:", json.dumps(event))

finding = event['detail']

threat\_type = finding['type']

severity = finding['severity']

ip = finding['service']['action'].get('remoteIpDetails', {}).get('ipAddressV4', 'Unknown')

log\_entry = {

'threatId': finding['id'],

'type': threat\_type,

'ip': ip,

'severity': str(severity),

'timestamp': datetime.utcnow().isoformat()

}

table.put\_item(Item=log\_entry)

if severity >= 7:

message = f"High severity threat detected:\nType: {threat\_type}\nIP: {ip}\nSeverity: {severity}"

sns.publish(TopicArn=SNS\_TOPIC\_ARN,Message=message, Subject='AWS GuardDuty Alert')

return {

'statusCode': 200,

'body': json.dumps('Threat processed successfully')

}

**10.2 IAM Policy Snippet for Lambda Role**

This sample IAM policy ensures that the Lambda Function has only the permissions necessary for its operation (DynamoDB write, SNS publish, CloudWatch logging).

Json:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"dynamodb:PutItem"

],

"Resource":"arn:aws:dynamodb:eu-north-1:123456789012:table/ThreatLogs"

},

{

"Effect": "Allow",

"Action": [

"sns:Publish"

],

"Resource":"arn:aws:sns:eu-north-1:123456789012:ThreatAlertTopic"

},

{

"Effect": "Allow",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": "\*"

}

]

}

**10.3 Sample DynamoDB Threat Logs**

Below are anonymised examples of threat entries stored in the DynamoDB table after Lambda execution.

Json:

{

"threatId": "ab23e456-7cde-4e45-b789-abc123def456",

"type": "Recon:PortProbe",

"ip": "198.51.100.23",

"severity": "8",

"timestamp": "2025-06-20T14:34:12Z"

}

{

"threatId": "cd89f012-3456-789a-bcde-f0123456789a",

"type": "UnauthorizedAccess:IAMUser/RootCredentialUsage",

"ip": "203.0.113.17",

"severity": "9",

"timestamp": "2025-06-21T09:45:22Z"

}

**10.4 AWS Services Used and Their Roles**

|  |  |  |
| --- | --- | --- |
| **Service** | **Description** | **Role in the Project** |
| Amazon GuardDuty | Threat detection service that analyses VPC, CloudTrail, DNS logs | Detects threats and triggers Lambda events |
| AWS Lambda | Serverless compute to process findings | Parses, logs, and triggers alerts |
| Amazon DynamoDB | NoSQL database with fast reads/writes | Stores processed threat logs |
| Amazon SNS | Notification service for alerts | Sends emails for high-severity threats |
| Amazon S3 | Object storage used for static website hosting | Hosts the frontend dashboard |
| Amazon CloudFront | CDN that catches and serves dashboard globally | Delivers the UI with reduced latency |
| IAM | Identity and Access Management | Controls access permissions for all services |
| CloudWatch | Monitoring and observability tool | Records Lambda performance, API metrics |

Table 10.1: AWS Services Used and Their Roles

**10.5 JSON: Sample GuardDuty Finding Structure**

To help understand the data structure passed to Lambda, here is a trimmed sample of a real GuardDuty finding event:

Json:

{

"detail": {

"id": "finding-01",

"type": "Recon:PortProbe",

"severity": 8,

"service": {

"action": {

"remoteIpDetails": {

"ipAddressV4": "192.0.2.0"

}

}

}

}

}

**REFERENCES**

1. Almorsy, M., Grundy, J., & Müller, I. (2016). An Analysis of the Cloud Computing Security Problem. Proceedings of the 2016 IEEE Asia Pacific Cloud Computing Congress (APCloudCC), 30–38. DOI: 10.1109/APCloudCC.2016.7586523
2. Sabahi, F. (2011). Cloud Computing Security Threats and Responses. 2011 IEEE 3rd International Conference on Communication Software and Networks, 245–249. DOI: 10.1109/ICCSN.2011.6014715
3. Zissis, D., & Lekkas, D. (2012). Addressing cloud computing security issues. Future Generation Computer Systems, 28(3), 583–592. https://doi.org/10.1016/j.future.2010.12.006
4. Amazon Web Services (AWS). (2023). Amazon GuardDuty – User Guide. Retrieved from https://docs.aws.amazon.com/guardduty/latest/ug/
5. Bhadauria, R., & Sanyal, S. (2012). Survey on Security Issues in Cloud Computing and Associated Mitigation Techniques. International Journal of Computer Applications, 47(18), 47–66. https://doi.org/10.5120/7269-0061