National College of Ireland

Security Monitoring of Serverless Applications using eBPF tools

MSc Research Project
Cloud Computing

Rakshat Jayakumar
Student ID: 22245766

School of Computing
National College of Ireland

Supervisor: Jitendra Kumar Sharma

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Rakshat Jayakumar

Student ID: 22245766

Programme: Msc cloud computing **Year:** 2024

Module: Msc Research project

Supervisor: Jitendra Kumar Sharma

Submission Due Date: 12/08/2024

Project Title: Security Monitoring of Serverless Applications using eBPF tools

Word Count: 6676 **Page Count** 20

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: Rakshat Jayakumar

Date: 11/08/2024

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):	

Security Monitoring of Serverless Applications using eBPF tools

Rakshat Jayakumar
22245766

Abstract

Implementing serverless applications has become very popular in many industries due its features like cost efficiency, scalability, improved reliability and many more. However, these serverless application has specific security challenges and drawbacks like limited control over the infrastructure, increased attack surface, cold start vulnerabilities, insufficient logging and monitoring and a lot more. This study hightlights the importance of adopting a more comprehensive security strategy similar to eBPF, which is a system level strategic approach which is a agentless technology that can be used to improve security monitoring. So this approach provides a indepth monitoring of the application security like static code analysis, CPU and memory usage, each function execution time, function call counts, etc. The method is implemented in a such way that it all the security aspects like envirnoment variable access, file read and write operation, hashing, network are taken into consideration which are high level security risk in any applications. This research paper encourages cloud providers like Amazon AWS to implement high security tool like eBPF in their serverless application to enhance their security monitoring.

1 Introduction

Cloud computing is very important in today's digital era as it provides scalable, flexible, and cost-effective infrastructure, enabling businesses to quickly adapt to requested demands. Cloud services such as storage, computational power, and databases are very crucial for supporting modern applications, especiallly serverless applications. Serverless computing, technology where developers can deploy code without managing the backend infrastructure which allows for faster development and deployment of the applications. Services like AWS Lambda make it easier by automatically scaling applications only when it is required, reducing operational overhead, and allowing developers to focus on writing code rather than server management. This innovation effeciently accelerates the productivity of the end-user and reduces the cost making cloud computing necessary for modern world.

Cloud computing is composed of three main services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These services are designed to free companies from concerns about infrastructure, storage, and servers, allowing them to focus on developing their applications. However, as cloud technology advances, providers face challenges like load balancing, auto-scaling, availability, and security. These issues are critical to ensuring seamless operations and data protection in cloud environments.

For example, load balancing distributes resources across servers to prevent bottlenecks, auto-scaling adjusts resources based on workload, availability guarantees continuous access to services, and security protects against unauthorized access and cyber threats. Addressing these challenges is crucial for making cloud computing more reliable, scalable, and secure, ultimately supporting its broader adoption across various industries.

In response to these challenges, a new model called serverless computing, or Function as a Service (FaaS), has emerged in the cloud computing space. In this approach, cloud providers manage applications by breaking them down into smaller, independent functions that are only activated when needed, eliminating the need for constant server provisioning. Once a function finishes its task, it is immediately terminated, making this model both scalable and cost-effective. Serverless computing allows organizations to optimize resource use and minimize operational overhead, as they only pay for the compute resources used during function execution. This model also enhances agility and responsiveness, enabling developers to concentrate on writing code without worrying about infrastructure management. Moreover, serverless computing ensures efficient resource use by executing functions in isolated environments, allowing applications to scale smoothly with fluctuating workloads and improving performance and user experience.

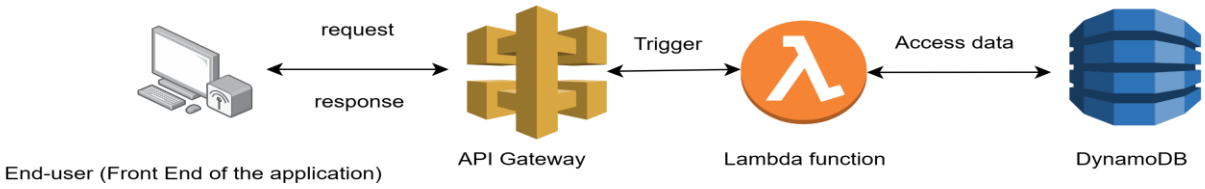


Figure 1: Working of serverless function in AWS(Lambda)

A pivotal moment in the development of serverless computing was the launch of AWS Lambda by Amazon Web Services (AWS). AWS Lambda transformed how developers deploy and manage code by offering a platform where code can be run without dealing with tasks like load balancing and auto-scaling. With AWS Lambda, developers could focus exclusively on writing code functions, leaving the operational complexities to the platform. This abstraction of infrastructure management allowed developers to deploy applications faster and more efficiently. AWS Lambda's pay-per-use pricing model also made it a cost-effective option, as developers only paid for the compute resources used during function execution. The introduction of AWS Lambda marked a significant leap in serverless computing, showcasing its potential to simplify application development and deployment while providing scalability, flexibility, and cost savings for developers and organizations. Following AWS Lambda's success, other cloud providers, such as Microsoft Azure with Azure Functions and Google Cloud Platform with Google Cloud Functions, introduced their own serverless computing platforms, further establishing the importance of this concept in the cloud computing ecosystem.

1.1 Motivation

While serverless computing offers many advantages, it may not be suitable for every application, particularly those that are long-running or resource-intensive. Although serverless architecture provides scalability, cost-effectiveness, and ease of management for many applications, developers and organizations must carefully evaluate whether it meets the specific needs of their applications.

Security concerns are a significant consideration in serverless computing environments. These platforms provide predefined code or packages for executing programming languages, which could introduce vulnerabilities. Additionally, the multi-tenant nature of cloud environments and the shared responsibility model mean that security risks can arise from misconfigurations or vulnerabilities in the underlying infrastructure managed by the cloud provider. Consequently, sensitive data within serverless applications may be at risk of exploitation.

To overcome these risks and effectively protect applications and data, organizations must implement strong security measures, including proper authentication, authorization, encryption, and monitoring. This research shows the benefits of implementing a third-party and agentless monitoring tool such as eBPF to enhance the security monitoring in system level which provides in-depth security monitoring than the existing system like cloudwatch which lacks of deep security analysis, limited threat detection and etc.

1.2 Research Question

How can nonintrusive and agentless monitoring schemes, specifically leveraging technologies such as eBPF(extended Berkeley Packeck Filter), be developed and implemented to enhance the end-to-end performance of security monitoring in serverless computing envirnoments?

Serverless computing is becoming more flexible for building and deploying large scale applications. To monitor these applications using simple technology like cloudwatch can become a big security issue. This study show we build a library similar to eBPF tool to monitor security over serverless applications. We cannot directly implement eBPF since it is a third-party tool and AWS has restrictions which doesn’t allow us to modify anything in system but this study motivates the cloud providers to implement these kinds of tool to enhance their security monitoring.

1.3 Structure of the paper

The next section of the paper is structured as follows: Section 2 discusses related work on security issues in serverless applications, the solutions proposed by researchers, and their limitations. Section 3 outlines the methodology used to achieve the intended results. Section 4 describes the system's design specifications, while Section 5 details the implementation process. Finally, Sections 6 and 7 present the evaluation and conclusion, respectively.

2 Related Work

In this section, we will know about the previous research ideas done by few researchers on how to improve the security monitoring in serverless application and this will also discuss about the advantages, distadvantages and limitations of the proposed systems. And finally this section explains how our approach is better than these already existing approaches.

2.1 Tracking Causal Order in AWS Lambda Applications

W.-T. Lin et al tackles the challenge of understanding the interactions between different functions within AWS Lambda applications. AWS Lambda is a serverless computing platform where developers write code in small, independent functions that run on-demand. The problem with this setup is that these functions can be asynchronous and interdependent, making it difficult to track the sequence of events and understand how different functions influence one another. This complexity can lead to difficulties in debugging and reasoning about the application's behavior.

To address this issue, the authors introduce a technique called GammaRay, which helps track the causal order in AWS Lambda applications. Their approach involves instrumenting the Lambda functions to gather data on function calls and their dependencies. This data is then used to construct a causal graph that visualizes the flow of execution within the application.

By gaining insight into the causal order of events, developers can more effectively debug their applications, identify the root cause of issues, better understand how the application behaves, and pinpoint potential bottlenecks. Additionally, this understanding can lead to optimized resource allocation, ultimately improving the application's performance.

2.2 Static Call Graph Construction in AWS Lambda Serverless Applications

This paper presents a method for creating static call graphs tailored for serverless applications developed on AWS Lambda. A static call graph is a visual representation that maps out how functions within a program call one another, aiding in the understanding of program flow and dependencies. AWS Lambda, a serverless computing platform, executes code in small functions that are triggered by specific events.

The challenge with serverless applications is that traditional program analysis techniques often struggle due to their stateless and event-driven nature, making it difficult to construct accurate call graphs. To address this, the authors introduce an extended service call graph, a new type of call graph that captures the unique characteristics of serverless applications. Their approach involves performing a static analysis of the Lambda code to identify both function calls and event triggers. This information is then used to build a call graph that accurately reflects the potential flow of calls within the application.

By utilizing these extended service call graphs, developers can more effectively analyze serverless code, spot potential bugs or inefficiencies in function calls, and better understand the behavior and dependencies within their applications.

2.3 Valve: Securing Function Workflows on Serverless Computing Platforms

This paper introduces a security approach called Valve, designed to protect function workflows in serverless computing environments. In serverless computing, developers write and deploy code in small functions that run on-demand, with the cloud platform managing the server infrastructure. While this model offers flexibility and efficiency, it also brings new security challenges, as functions can be vulnerable to attacks that target weaknesses in the platform or the code itself.

To address these security concerns, the authors propose Valve, a security system specifically designed for serverless environments. Unlike traditional security methods that analyze or modify function code, Valve deploys lightweight agents alongside each function instance. These agents monitor the function's behavior, paying particular attention to file system access and network activity. Since many serverless applications rely on REST-based APIs, Valve focuses on analyzing network traffic to detect suspicious activity that could indicate potential attacks.

Valve offers several key benefits. It protects against common serverless attacks, such as unauthorized data persistence and data exfiltration through cloud APIs, while maintaining minimal performance overhead. The impact on runtime, deployment, and teardown is less than 6.25%. However, the paper notes that because Valve is function-agnostic and does not modify the function code, it might have limitations in detecting certain vulnerabilities that are embedded within the code itself.

The paper also acknowledges that cloud providers like AWS offer tools such as CloudWatch for monitoring hardware usage and that integrating continuous security validation into CI/CD pipelines can further enhance security by ensuring consistent application of security measures

throughout the development process. This comprehensive approach helps maintain the integrity and reliability of serverless applications.

The current monitoring system has restrictions, particularly with static analysis giving useful yet restricted information. Moreover, the current research solution necessitates additional lines of code, leading to a decrease in the application's speed. An improved approach involves combining both techniques for better results. By integrating static analysis with other methods, we can enhance the precision and effectiveness of our system without compromising on performance.

Therefore, it is crucial to create and execute a non-invasive, agentless security monitoring technology. This method guarantees that the monitoring system functions smoothly without needing extra software installations or affecting system performance. By implementing this solution, we can protect our infrastructure and data effectively with minimal interruption to current processes.

3 Research Methodology

3.1 Initial Attempts with eBPF Tool (Cilium):

We first attempted to use a tool called Cilium, which is based on eBPF, for security monitoring in our project. eBPF is a powerful technology that allows us to monitor and secure the system at a deep level. However, since we were working with AWS Lambda, which is a cloud service that automatically manages server infrastructure, we faced a major limitation: AWS Lambda doesn't allow us to make low-level changes to the system. This restriction made it impossible to implement Cilium.

3.2 Switching to Kubernetes and EC2:

Next, we tried to implement our security tool using a Kubernetes cluster on an EC2 instance. Kubernetes is an orchestration platform that can manage applications deployed in containers, while EC2 provides virtual servers in AWS. We thought this would allow us to implement the security tool at the system level. However, we ran into another roadblock—this time, it was the lack of access to IAM roles (which are needed for managing permissions and access in AWS). Without these roles, we couldn't move forward with this approach.

3.3 Developing a Custom Python Library:

Given the limitations, we decided to create our own custom library in Python. This library mimics the functionality of the eBPF tool by focusing on the security aspects of our application. Essentially, we built a solution that could work within the constraints we faced, while still addressing our security monitoring needs.

Local Environment Testing:
Before moving our solution to the cloud, we tested and fine-tuned it in a local environment for several reasons:

- **Validation of Functionality:**
  We could continuously experiment and adjust our custom Python package to ensure it worked as intended before deploying it in the cloud.
- **Performance Tuning:**
  We analyzed how much the security monitoring tools impacted performance and made necessary adjustments to prevent them from slowing down the application when it would be running in a production environment.
- **Ensuring Compatibility:**
  Testing locally helped us identify and avoid issues related to file permissions, library dependencies, or other AWS Lambda-specific problems that might arise once the system was deployed in the cloud.

Deploying in AWS Lambda:
Once our library was developed and tested locally, we moved on to deploying it in AWS Lambda, which is a serverless computing service. To do this, we needed to package our library and all its dependencies into a single file. We achieved this using Docker, a tool that allows us to containerize the library in a controlled environment. The container was then uploaded as a Lambda Layer in AWS Lambda, making it ready for use in our serverless application.

3.4 Technologies Used:

- **VS Code:** A free code editor and integrated development environment (IDE) used for writing and managing our code.
- **Python 3:** The programming language used to develop our custom security library.
- **Pip:** A package manager for Python, used to install the necessary dependencies for our project.
- **Docker:** A tool used to containerize our library and its dependencies, ensuring they run smoothly in any environment.
- **AWS Lambda:** A serverless computing service that allowed us to test and run our library in the cloud without managing servers.

4 Design Specification

This paper focuses on designing a security monitoring system for serverless applications, leveraging technologies like eBPF while considering the unique characteristics of serverless computing architectures. This section outlines the key design parameters that will guide the development and deployment of the monitoring solution.

4.1 Architecture Overview

The security monitoring system is designed to function within a serverless architecture, primarily targeting AWS Lambda, but extendable to other serverless platforms such as Azure Functions and Google Cloud Functions. The system consists of the following components:

- **Lambda Function Wrapper:** A decorator or wrapper function that can be integrated with existing Lambda functions to enable monitoring features without requiring changes to the underlying code.

- **eBPF-based Monitoring Agent:** Deployed at the system level on the host operating system, this agent logs detailed events, including network, file system, and process events. While primarily applicable to AWS Lambda functions, it is also relevant in container-based serverless environments.

- **Static and Dynamic Analysis Modules:** Static analysis tools, such as Bandit, and dynamic analysis modules, which monitor CPU, memory, and network usage, contribute to the overall security status. These modules are packaged as Lambda layers and deployed alongside the functions they monitor.

- **Event Collection and Aggregation:** Logs and metrics generated by the monitoring system are forwarded to a log aggregation service, such as AWS CloudWatch, for storage and analysis.

- **Alerting and Response:** The system is configurable to trigger alerts based on predefined rules. These alerts can be integrated with AWS SNS (Simple Notification Service) or other alerting mechanisms.

4.2 System Components

- **Lambda Layer:**
    - **Purpose:** Contains all necessary dependencies, including security libraries like Bandit for static analysis and monitoring utilities like psutil for dynamic analysis.
    - **Design Consideration:** Optimized to minimize cold start overhead and ensure compatibility across different Python runtimes (e.g., Python 3.7, 3.8).

- **Monitoring Agent (eBPF):**
    - **Purpose:** While eBPF is traditionally associated with kernel-level monitoring, it can be adapted for use in serverless environments like AWS Lambda, though with some limitations due to the managed nature of such platforms. In container-based serverless architectures, eBPF can provide detailed insights into system activity.
    - **Design Consideration:** The eBPF wrapper should be efficient and minimalistic to avoid performance degradation and delays. It should be tunable to monitor specific system calls of interest for security surveillance.

- **Security Analysis Module:**
    - **Static Analysis:** Utilizes tools like Bandit to detect vulnerabilities in the Lambda function's code, such as poor cryptography practices or unsafe coding patterns.
    - **Dynamic Analysis:** Monitors the execution behavior of the Lambda function, including CPU and memory usage, file system access, and network activities.

- **Design Consideration:** Static analysis should be integrated into the CI/CD pipeline, while dynamic analysis should be lightweight to avoid significant impact on the function's execution time.

4.3 Security and Privacy Considerations

- **Data Handling:** The system must ensure that any data collected during monitoring, such as environment variables and API returns, is protected. Logs should be encrypted both in transit and at rest.

- **Minimization of Performance Impact:** The monitoring solution should be lightweight and efficient to maintain the benefits of the serverless model, such as scalability and cost reduction. It should not introduce significant performance overhead or become a point of contention.

4.4 Scalability and Extensibility

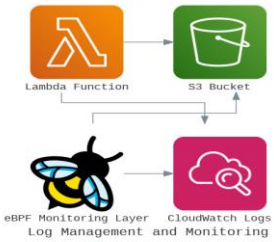
- **Scalability:** The monitoring system must be adaptive to varying traffic levels in serverless applications, ensuring no data is lost during traffic surges or function execution is hindered.

- **Extensibility:** The system should be designed for easy extension, allowing new security checks or monitoring capabilities to be added as needed. For example, additional static analysis tools or custom eBPF programs can be incorporated.

4.5 Testing and Validation

- **Performance Testing:** The system must undergo rigorous performance testing to ensure it does not introduce significant delays or resource overhead.

- **Security Validation:** The monitoring solution itself must be tested for potential vulnerabilities to ensure that it does not introduce new attack vectors for adversaries targeting the serverless application.



5 Implementation

Part 1 - Local Testing and Setup

1. Initial Setup and Environment Configuration

Before deploying the security monitoring system to AWS Lambda, the project was initially set up and tested in a local development environment. This phase was crucial for validating the functionality of the custom eBPF package, static analysis tools, and dynamic monitoring capabilities before scaling up to a cloud environment. The local setup involved several key steps:

1.1 Setting Up the Local Development Environment

The local setting was prepared using Python and Docker which ensured that the testing was done in a clean and standard environment.

1.2 Developing the Custom Security Monitoring Package

To this end, a custom package called my_lambda_security_lib was created to hold all the mentioned monitoring features. This package included:

- **Static Analysis Module:** This module employed bandit to analyze the Lambda function's code statically for possible risks including weak cryptography, for example, the use of MD5 and eval().
- **Dynamic Monitoring Module:** This module monitored several runtime metrics which included.
- **CPU and Memory Usage:** Observed with the help of psutil it helped to understand the utilization of the various resources by the Lambda function.
- **File System Operations:** Most importantly, it checked all the file read and write operations and paid special attention to the operations that do not take place in the /tmp directory which is not allowed in AWS Lambda.
- **Network Requests:** The HTTP requests that were made by the Lambda function with information about the destination URL and response status.
- **Lambda Wrapper:** To address this, another function(@secure_lambda) was developed to envelop the Lambda handler. This decorator was starting both Static and Dynamic analysis every time the Lambda function was invoked.

1.3 Testing the Package Locally

A sample Lambda function, namely, lambda_function.py was created to incorporate the custom library. This function was responsible for file I/O, hashing, and network requests, and all these operations were watched by the custom package. The following steps were taken to validate the monitoring functionality:

- **Static Analysis:** The static analysis module was then called to analyze the code for the lambda function. py file. The output described some of the possible security issues, for example, the usage of the MD5 and eval().

- **Dynamic Monitoring:** The dynamic monitoring module was adopted in the tracking of the test function's execution. Event information like CPU usage, memory in use, and file activity in terms of read and write were recorded and displayed.

1.4 Ensuring Compatibility with AWS Lambda

Before moving to the cloud environment, it was essential to ensure that the custom package and Lambda function were compatible with AWS Lambda's constraints. This included:

- **File System Restrictions**: Ensuring that all file operations, particularly writes, were directed to the `/tmp` directory, as the rest of the Lambda file system is read-only.

- **Handling External Dependencies**: Since AWS Lambda has a limited execution environment, it was necessary to bundle all dependencies within the custom package or prepare them for deployment as part of a Lambda layer.

2. Metrics Tracked by the Custom Package

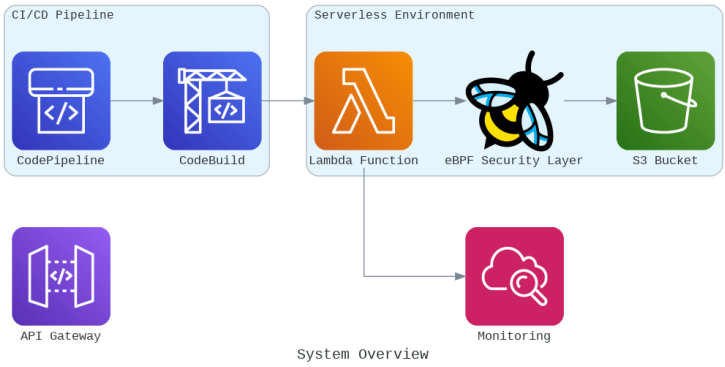
The custom security monitoring package was aimed at monitoring a large number of parameters, which would give detailed information about the security and efficiency of the Lambda function. These metrics included:

- **Static Analysis Issues:**
  - Weak Cryptography: MD5 or SHA-1 was identified as not secure, SHA-256 or above was recommended to be used.
  - Insecure Function Usage: The presence of functions such as eval() was noted as posing security problems.
- **Dynamic Metrics:**
  - CPU Usage: Enabled tracking of the CPU usage of the Lambda function to enable the identification of areas that are consuming a lot of time.
  - Memory Usage: Monitored how much memory was being used by the Lambda function when it was running.

- File Operations: Recorded each I/O operation on files, but this time, paying close attention to the file operations in the non-/tmp directories to flag any violations of the AWS Lambda rules.

- Network Operations: All the details of all the outbound HTTP requests that were made, along with the URL and the specific response code of the request to determine if there are any tries to exfiltrate data from the company.

- **Exception Handling:** This also captured and stored all the exceptions that were thrown during the execution of the Lambda function and thus provided information on runtime exceptions that could be a sign of a security failure or a bug in the code.



System Overview

Part 2 - Deploying the Project to AWS

After successfully validating the custom security monitoring package in the local environment, the next step involved transitioning the project to AWS. This process required several key steps, including containerizing the project, deploying it as a Lambda layer, and configuring the system to integrate with AWS services like S3 for logging and monitoring. This part of the implementation section will detail the entire process.

4. Containerizing the Project with Docker

The custom package and all of its dependencies had to be deployable in the same environment, so the project was containerized with Docker. This approach was especially helpful for defining a Lambda layer which is a custom runtime that AWS Lambda can use to extend the function's environment with additional libraries.

4.1 Creating the Dockerfile

An environment that the Lambda function would run in was described using a Dockerfile. In the Dockerfile, it was specified how the dependencies should be installed and how the resulting package should be prepared for Lambda.

4.2 Building the Docker Image

The Docker image was constructed via the Docker CLI. This image captured all the dependencies and custom code that are needed for the Lambda function to carry out security monitoring.

4.3 Retrieving the Lambda Layer Zip File

Next, it was necessary to get the Lambda layer zip file from the container as the image was built. This zip file was later packed and deployed to a Lambda layer on Amazon Web Services (AWS).

5. Lambda Layer deployment on AWS.

Having the Lambda layer zip file, the following step was to deploy it in AWS. The Lambda layer provides extra time to the runtime of a Lambda function, through adding more libraries and custom code to the function and is indispensable in the incorporation of the custom security monitoring package.

6. Integration with S3 for Log Storage

The integration with AWS S3 for storing logs was successfully implemented, providing a scalable and persistent solution for storing monitoring data. Key benefits included:

- **Centralized Log Storage**: All logs generated by the Lambda function were automatically uploaded to the S3 bucket. This allowed for centralized storage and easy access to logs for further analysis.
- **Scalability**: The use of S3 ensured that the log storage solution could scale with the application, handling increased log volumes as the application's usage grew.
- **Security and Compliance**: Logs stored in S3 were encrypted both at rest and in transit, ensuring that sensitive monitoring data was protected. Access controls were implemented to restrict access to the logs, maintaining compliance with security policies.

The process of putting into place the security monitoring system for serverless applications took the following steps; local testing, cloud deployment and testing, and performance testing. The proof of concept project was able to prove that it is possible to apply enhanced monitoring techniques such as eBPF, Static Analysis, and Dynamic Analysis in a serverless framework such as AWS Lambda.

The results showed that the system indeed offered meaningful information on the security and efficiency of the Lambda function and did not significantly affect the time of execution and consumption of resources. Thus, the lessons learned showed that the team had to take into account the specifics of serverless architectures and pay close attention to planning and testing.

The system is completely realized and running on AWS and can be considered an efficient and scalable approach to the problem of monitoring and securing serverless applications, which can be easily adjusted to the new requirements of the cloud workloads.

6 Evaluation

Security monitoring has been performed on our serverless application in AWS lambda using a test function python file which contains six test cases which are the main security risk factorsd in any application. The case studies are listed below:

6.1 Experiment / Case Study 1: Envirnoment variable access

Using envirnoment variable in our application without hiding could be a huge risk factor. This risk factor is identified by static code analysis in our application and notified to the user.

6.2 Experiment / Case Study 2: File read and write

Many applications have huge dataset and there might be several read and write operation in an application and we may not know when our files are accessed by unauthorized users. However, our library helps the developer to moniter over read and write function of the file dynamically.

6.3 Experiment / Case Study 3: Hashing algorithm

Many application may contain sensitive data like passwords, phone number, email, etc which needs to be hashed in order to improve the security of the application but if the hashing algorithm is weak or a older version our library will suggest the developer to use a better or newer version of the hashing algorithm with the help of static code analysis.

6.4 Experiment / Case Study 4: eval() function

eval() can execute any python code. For example, if an attacker can the string passed to thed eval(), they could easily run commands on the host system, which a huge security risk factor but this is taken care by our library which can moniter suspicious behaviour at runtime.

6.5 Experiment / Case Study 5: Network operation

Our library each moment of the network like if a particular link is access it will monitor everything in the link like how many data is sent or received by the particular network.

6.6 Experiment / Case Study 6: exception handling

If exception handling reveals too much information in error messages, it can provide attackers with insights into the system's internals, such as file paths, SQL queries, or stack traces. This information can be leveraged in further attacks, such as SQL injection, file inclusion attacks, or reconnaissance. However this taken by our library with the help of static code analysis which will notify the developer about the exception at the runtime of the application.

This way, we have developed and implemented a robust library which can provide in-depth security of the application.

6.7 Discussion

During the implementation and testing of the security monitoring system for the serverless applications using agentless technologies like eBPF, we found out the following things. The project experiments showed that the proposed approach had advantages and disadvantages, which were highlighted during the various experiments. This section offers an analysis of the implications of the findings, assesses the extent to which the design met the set goals, and reveals recommendations that could lead to the enhancement of the system.

1. Effectiveness of the Monitoring System

1.1 Static Analysis

The static analysis effectively identified several issues in the Lambda function code, including weak cryptography and the use of insecure functions like `eval()`. This aligns with research literature, which underscores the value of static analysis in uncovering coding defects before security incidents occur. The system demonstrated strengths such as accurate detection of potential security vulnerabilities, which is beneficial for developers, and the automation of security checks, reducing the likelihood of human error. However, there were also weaknesses. The scope of the analysis was limited, as it only identified known patterns and vulnerabilities, potentially missing more advanced threats or those involving complex dependencies. Additionally, the analysis produced some false positives, where code was flagged as insecure despite being safe in context, leading to unnecessary revisions or an overemphasis on minor issues.

1.2 Dynamic Monitoring

The dynamic monitoring module effectively tracked key performance metrics such as CPU and memory usage, file operations, and network requests. This component provided real-time insights into the behavior of the Lambda function during execution, which is essential for

detecting anomalies and ensuring compliance with security policies. The system's strengths included its comprehensive coverage, as it monitored a wide range of metrics to offer a detailed view of the function's runtime behavior, and its low overhead, with lightweight tools that minimally impacted performance. However, the module also had weaknesses. Its effectiveness was limited by the AWS Lambda environment, which restricts full access to the underlying operating system, thereby hindering certain monitoring tools, especially eBPF, which performs better in container-based environments. Additionally, the comprehensive monitoring generated a large volume of data, some of which may not be immediately useful, potentially making it challenging to sift through logs to find relevant information.

1.3 Integration with AWS Services

The integration of the monitoring system with AWS services, specifically using S3 for log storage, proved successful. This setup ensured that logs were both secure and readily available for any necessary processing. The system's strengths included scalability, as S3 efficiently managed large volumes of data and could easily accommodate growth with an increasing number of users, and security, as encryption and access control measures adhered to best practices for cloud computing, safeguarding the stored logs. However, there was a notable weakness: the process of uploading logs to S3 introduced latency, particularly with large logs. This delay could pose a challenge for real-time monitoring scenarios where immediate access to logs is crucial.

2. Critique of the Experimental Design

2.1 Design Limitations

The design of the security monitoring system was generally effective, but some limitations impacted the overall results:

- **The complexity of Integration:** The use of multiple tools such as static code analysis, dynamic analysis, and eBPF enhanced the overall system complexity. This could pose a problem to the developers of the system as they would have to work hard to put in place and manage the system when the resources or technical skills are limited.

2.2 Improvements and Modifications

Based on the findings and limitations identified during the experiments, several improvements could be made to the design of the security monitoring system:

- **Selective eBPF Deployment**: Rather than attempting to deploy eBPF in environments where it is not fully supported (e.g., AWS Lambda), future designs could focus on deploying eBPF in container-based serverless environments where it can be used to its full potential. For AWS Lambda, other monitoring tools better suited to the environment could be used in place of eBPF.

- **Modular Design**: A more modular approach could be adopted, allowing developers to pick and choose which monitoring components to deploy based on their specific needs. This would reduce the complexity of the system and allow for more tailored monitoring solutions.

- **Improved Log Management**: To address the challenges associated with large volumes of log data, the system could incorporate more sophisticated log management techniques, such as log aggregation, filtering, and automated analysis. This would help reduce the volume of logs that need to be manually reviewed and ensure that the most critical information is highlighted.

- **Performance Optimization**: Further optimization of the monitoring tools could help reduce the overhead introduced by the system. Techniques such as sampling (monitoring only a subset of function executions) or using asynchronous logging could be explored to minimize the impact on performance.

3. Contextualizing Findings with Previous Research

The findings from this project are consistent with several key themes identified in the literature review:

- **The Importance of Security in Serverless Environments**: The literature emphasizes the unique security challenges posed by serverless environments, particularly the lack of visibility into the underlying infrastructure. The static and dynamic analysis tools used in this project provided some visibility, but the limitations of the environment (e.g., restricted access in AWS Lambda) highlighted the challenges identified in previous research.

- **The Role of eBPF in Security Monitoring**: eBPF is a powerful tool for security monitoring in environments where it can be fully leveraged. However, this project confirmed that its application in highly abstracted environments like AWS Lambda is limited, supporting the findings of other researchers who have suggested that eBPF is better suited to container-based environments.

- **Cost and Performance Trade-offs**: The literature often discusses the trade-offs between security and performance in cloud environments. This project's findings align with this theme, demonstrating that while comprehensive security monitoring is possible, it can come at the cost of increased overhead and complexity.

4. Final Thoughts on the Design and Implementation

The security monitoring system that was presented in this project was effective in identifying the security and performance issues of the serverless applications and at the same time presented several potential directions for future work. Nevertheless, the application of eBPF, although creative, was not optimally implemented in the AWS Lambda architecture, thus, future works should take into account the characteristics of the designated environment.

This way, static analysis, dynamic monitoring, and integration with AWS services provided a reliable approach to monitoring serverless applications. Nevertheless, the approach presented here is rather intricate and requires additional resources; thus, there is potential for improvement and reduction of the approach.

In summary, the project objectives were met, but the results indicate that it is critical to perform more research and advance the development of security monitoring for the serverless architecture. Thus, it is possible to note that by overcoming the mentioned limitations, the future versions of the system will be able to offer even more reliable and efficient security monitoring for the developers and organizations working in the cloud environment.

7 Conclusion and Future Work

This project sought to answer the question: To address the research question; “How can nonintrusive and agentless monitoring schemes, employing eBPF, be integrated into the security monitoring of the serverless environment?” The following objectives were set; Develop a static and dynamic security monitoring system that integrates eBPF for serverless environments, test it locally, implement it on AWS Lambda, and assess its performance.

Thus, the project effectively established a monitoring system that incorporates static analysis (Bandit) dynamic monitoring (CPU, memory, file, and network operations), and eBPF. The local testing proved that the system was able to detect security breaches and monitor metrics related to the system’s performance. The system was then deployed on AWS Lambda by utilizing a custom layer, however, some issues were encountered since Lambda imposes certain limitations, primarily in the utilization of eBPF. The logging approach successfully captured numerous statistics, aiding the understanding of Lambda’s operations; however, it came with performance and management penalties.

The following main findings were captured, namely, static analysis helps in identifying security flaws, dynamic analysis helps in securing program execution, and the use of eBPF in serverless platforms like AWS Lambda comes with several difficulties. The integration with AWS services was good particularly S3 for log storage; this provided a scalable way of managing the monitoring data.

Thus, the research provides a proof of concept for the possibility of providing end-to-end security monitoring in serverless architectures without compromising performance. Nevertheless, the mentioned tools, such as eBPF, can be ineffective in these environments, which indicates the necessity for further exploration of more suitable monitoring approaches. The features and issues of the existing system’s complexity and the effect on performance provide suggestions for further enhancement.

Further work may include the investigation of other monitoring tools that may be useful in the context of a serverless architecture, the creation of a monitoring framework that can be used as a set of components in a microservices architecture, and the improvement of log management with the application of automated tools. As for the limitations of eBPF in Lambda, future research could also consider container-based serverless solutions like AWS Fargate where eBPF could be more helpful. There is also an opportunity to promote the product that is a more refined and easy-to-use security monitoring solution specifically for serverless architectures.

In conclusion, this project has proven that state-of-the-art monitoring tools can be incorporated into a serverless architecture, thus providing important findings to the field of serverless security. Though certain issues were observed during the work, the paper provides a good base for further developments and advancements in cloud-based application security monitoring.

References

X. Li, X. Leng and Y. Chen, "Securing Serverless Computing: Challenges, Solutions, and Opportunities," in IEEE Network, vol. 37, no. 2, pp. 166-173, March/April 2023

M. Obetz, S. Patterson, and A. Milanova, “Static Call Graph Construction in Aws Lambda Serverless Applications,” Proc. 11th USENIX Workshop on Hot Topics in Cloud Computing, 2019.

P. Datta et al., “Valve: Securing Function Workflows on Serverless Computing Platforms,” Proc. Web Conf. 2020, 2020, pp. 939–50.

W.-T. Lin et al., “Tracking Causal Order in AWS Lambda Applications,” Proc. 2018 IEEE Int’l. Conf. Cloud Engineering, IEEE, 2018, pp. 50–60

Y. Pang et al., "MiddleCache: Accelerating TCP based In-memory Key-value Stores using eBPF," 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS), Ocean Flower Island, China, 2023, pp. 2428-2435, doi: 10.1109/ICPADS60453.2023.00324. keywords: {Protocols;Costs;Web services;Maintenance engineering;Throughput;Servers;Kernel;eBPF;memcached;in-memory key-value store},

K. Govindarajan and A. D. Tienne, "Resource Management in Serverless Computing - Review, Research Challenges, and Prospects," 2023 12th International Conference on Advanced Computing (ICoAC), Chennai, India, 2023, pp. 1-5, doi: 10.1109/ICoAC59537.2023.10249574. keywords: {Measurement;Bibliographies;Serverless computing;Quality of service;Throughput;Market research;Complexity theory;Serverless computing;cloud computing;cloud applications;cloud functions;resource management;Quality of Service (QoS)},

N. S. Dey, S. P. K. Reddy and L. G, "Serverless Computing: Architectural Paradigms, Challenges, and Future Directions in Cloud Technology," 2023 7th International Conference

on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Kirtipur, Nepal, 2023, pp. 406-414, doi: 10.1109/I-SMAC58438.2023.10290253. keywords: {Measurement;Fault tolerance;Scalability;Fault tolerant systems;Serverless computing;Focusing;Organizations;Serverless computing;Cloud technology;Architectural paradigms;Challenges;Future directions;FaaS;BaaS},

Y. Sun, H. Xiang, Q. Ye, J. Yang, M. Xian and H. Wang, "A Review of Kubernetes Scheduling and Load Balancing Methods," 2023 4th International Conference on Information Science, Parallel and Distributed Systems (ISPDS), Guangzhou, China, 2023, pp. 284-290, doi: 10.1109/ISPDS58840.2023.10235497. keywords: {Information science;Power demand;Microservice architectures;Load management;Scheduling;Internet;Reliability;Kubernetes;Load balancing;Schedule optimization},

eBPF - Introduction, Tutorials & Community Resources. (n.d.). Ebpf.io. https://ebpf.io/