Configuration Manual

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
Lecturer:	Jitendra Kumar Sharma
Submission Due Date:	12/08/2024
Project Title:	Security Monitoring of Serverless Applications using eBPF tools
Word Count:	1603 **Page Count:** 13

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: 12/08/2024

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):	

Configuration Manual

Rakshat Jayakumar
Student ID: 22245766

1 Introduction

This research project has been implemented in two ways. The first way is test the library locally. The project was developed and tested in a local development envirnoment with help of Visual Studio Code and later on it deployed to Amazon AWS with the help of docker.

1.1 Prerequisite

Before we begin to build the project we must have the developer tools installed in our systems.

- Visual Studio Code: Free lightweight code editior.
- Python: A Programming language that has been used to develop the custom security monitoring package together with the Lambda function.
- Pip: python package manager to install required python libraries.
- Docker: A tool for running applications in a container and then for the emulation of the AWS Lambda environment on the local machine.
- AWS account: Platform which is used to create lambda functions.

2 Locally deploying the file

2.1 Developing the library

First we need to create a python package which needs to created to create monitoring functions like shown below:

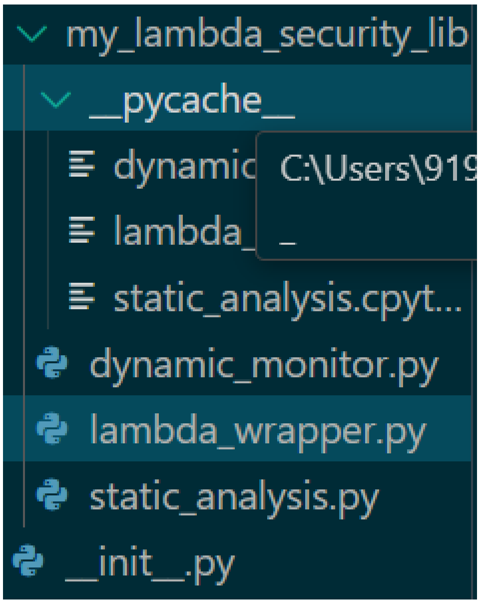


Figure 1: Structure of the package

In this case, I have created a package called “my_lambda_security_lib” which contains python files which are specifically designed to perform security operations.

```
import sys
import time
import os
import psutil
import threading
import requests

# Global variables to store metrics
execution_times = {}
function_call_count = {}
cpu_usage = []
memory_usage = []
network_requests = []
file_operations = []
resource_monitoring_active = True
start_time = None
end_time = None
call_depth = 0
max_call_depth = 0

def trace_calls(frame, event, arg):
    """Trace function calls and return time spent in each function."""
    global call_depth, max_call_depth
    if event == 'call':
        function_name = frame.f_code.co_name
        execution_times[function_name] = time.perf_counter()
        function_call_count[function_name] =
function_call_count.get(function_name, 0) + 1
```

```
        call_depth += 1
        max_call_depth = max(max_call_depth, call_depth)
        print(f"Function call: {function_name}, Current depth: {call_depth}")
    elif event == 'return':
        function_name = frame.f_code.co_name
        if function_name in execution_times:
            elapsed_time = time.perf_counter() -
execution_times[function_name]
            execution_times[function_name] = elapsed_time
            print(f"Function return: {function_name}, Time taken:
{elapsed_time:.6f} seconds")
        call_depth -= 1
    return trace_calls

def monitor_resources():
    """Monitor CPU and memory usage during the function execution."""
    global resource_monitoring_active
    process = psutil.Process(os.getpid())  # Get the current process

    # Initial baseline call to set up the monitoring
    process.cpu_percent(interval=None)

    while resource_monitoring_active:
        cpu_percent = process.cpu_percent(interval=None)  # Get the CPU usage
since the last call
        memory_info = process.memory_percent()
        cpu_usage.append(cpu_percent)
        memory_usage.append(memory_info)
        print(f"CPU Usage: {cpu_percent:.2f}%, Memory Usage:
{memory_info:.2f}%")
        time.sleep(0)  # Yield thread, but immediately resume to capture
continuous data

def start_dynamic_monitoring():
    global start_time
    print("Starting dynamic monitoring...")
    start_time = time.time()  # Record the start time

    # Start resource monitoring in a separate thread
    resource_monitor_thread = threading.Thread(target=monitor_resources)
    resource_monitor_thread.daemon = True
    resource_monitor_thread.start()

    # Start tracing function calls
    sys.settrace(trace_calls)

def stop_dynamic_monitoring():
    global resource_monitoring_active, end_time
    sys.settrace(None)
```

```
    resource_monitoring_active = False  # Stop resource monitoring
    end_time = time.time()  # Record the end time
    print("Dynamic monitoring stopped.")

    # Output the detailed summary
    output_detailed_summary()

def monitor_network_calls():
    """Monkey-patch the requests module to log all network requests."""
    original_get = requests.get

    def patched_get(*args, **kwargs):
        request_info = {
            'url': args[0],
            'method': 'GET',
            'status_code': None,
            'data_sent': len(kwargs.get('data', b'')),
            'data_received': 0
        }
        response = original_get(*args, **kwargs)
        request_info['status_code'] = response.status_code
        request_info['data_received'] = len(response.content)
        network_requests.append(request_info)
        print(f"Network request made to: {args[0]}, Status:
{response.status_code}")
        return response

    requests.get = patched_get

def monitor_file_operations():
    """Monitor file operations by monkey-patching the built-in open
function."""
    original_open = open

    def patched_open(*args, **kwargs):
        file_info = {
            'file_name': args[0],
            'mode': kwargs.get('mode', 'r')
        }
        file_operations.append(file_info)
        print(f"File operation: open, File: {args[0]}, Mode:
{file_info['mode']}")
        return original_open(*args, **kwargs)

    __builtins__['open'] = patched_open

def output_detailed_summary():
    """Output a detailed summary of all the collected metrics."""
    print("\n--- Detailed Summary ---")
```

```
    # Total execution time
    total_execution_time = end_time - start_time
    print(f"Total Execution Time: {total_execution_time:.6f} seconds")

    # Function execution times
    print("\nFunction Execution Times:")
    for function_name, elapsed_time in execution_times.items():
        print(f"- {function_name}: {elapsed_time:.6f} seconds")

    # Function call counts
    print("\nFunction Call Counts:")
    for function_name, count in function_call_count.items():
        print(f"- {function_name}: {count} calls")

    # Maximum call depth
    print(f"\nMaximum Function Call Depth: {max_call_depth}")

    # CPU usage stats
    if cpu_usage:
        avg_cpu_usage = sum(cpu_usage) / len(cpu_usage)
        peak_cpu_usage = max(cpu_usage)
    else:
        avg_cpu_usage = peak_cpu_usage = 0
    print(f"\nAverage CPU Usage: {avg_cpu_usage:.2f}%")
    print(f"Peak CPU Usage: {peak_cpu_usage:.2f}%")

    # Memory usage stats
    if memory_usage:
        avg_memory_usage = sum(memory_usage) / len(memory_usage)
        peak_memory_usage = max(memory_usage)
    else:
        avg_memory_usage = peak_memory_usage = 0
    print(f"Average Memory Usage: {avg_memory_usage:.2f}%")
    print(f"Peak Memory Usage: {peak_memory_usage:.2f}%")

    # Network requests
    total_data_sent = sum(req['data_sent'] for req in network_requests)
    total_data_received = sum(req['data_received'] for req in
network_requests)
    print("\nNetwork Requests:")
    if network_requests:
        for req in network_requests:
            print(f"- URL: {req['url']}, Method: {req['method']}, Status:
{req['status_code']}, Data Sent: {req['data_sent']} bytes, Data Received:
{req['data_received']} bytes")
    else:
        print("- No network requests made.")
    print(f"Total Data Sent: {total_data_sent} bytes")
```

```
    print(f"Total Data Received: {total_data_received} bytes")

    # File operations
    print("\nFile Operations:")
    if file_operations:
        for file_op in file_operations:
            print(f"- File: {file_op['file_name']}, Mode: {file_op['mode']}")
    else:
        print("- No file operations performed.")

    print("\n--- End of Summary ---")

# Apply additional monitoring
monitor_network_calls()
monitor_file_operations()
```

This code is used for dynamic monitoring which monitors several runtime metrics like total,execution time, function execution time, file operations and many more.

```
from .static_analysis import run_static_analysis
from .dynamic_monitor import start_dynamic_monitoring, stop_dynamic_monitoring

def secure_lambda(lambda_handler):
    def wrapper(event, context):
        # Run static analysis first
        issues = run_static_analysis()
        if issues:
            print("Static Analysis Issues Detected:", issues)

        # Start dynamic monitoring after static analysis
        start_dynamic_monitoring()

        try:
            # Execute the original lambda function
            result = lambda_handler(event, context)
        finally:
            # Stop dynamic monitoring after the function execution
            stop_dynamic_monitoring()

        return result

    return wrapper
```

This is lambda_wrapper.py which is used to run static code analysis and dynamic code analysis each time the lambda fuction is invoked.

```
import os
import subprocess
```

```
def run_bandit_analysis(target_directory):
    """
    Run Bandit security analysis on the specified directory and return a list
of issues.
    Bandit is a tool designed to find common security issues in Python code.
    """
    try:
        result = subprocess.run(
            ["bandit", "-r", target_directory],
            capture_output=True,
            text=True
        )
        return parse_bandit_output(result.stdout)
    except Exception as e:
        print(f"Error running Bandit analysis: {e}")
        return []

def parse_bandit_output(output):
    """
    Parse the output of Bandit to extract the relevant security issues.
    """
    issues = []
    lines = output.splitlines()
    for line in lines:
        if "Issue:" in line:
            issues.append(line.strip())
    return issues

def run_custom_static_checks(lambda_code_path):
    """
    Run additional custom static checks that are not covered by Bandit.
    """
    # Example check for weak cryptography (just a simple keyword search for
demonstration)
    issues = []
    with open(lambda_code_path, "r") as code_file:
        code_content = code_file.read()
        if "md5" in code_content or "sha1" in code_content:
            issues.append("Use of weak cryptography detected (MD5/SHA1).
Consider using SHA256 or higher.")
    return issues

def run_static_analysis():
    """
    Run the complete static analysis by combining Bandit and custom checks.
    """
    current_dir = os.path.dirname(os.path.abspath(__file__))
    lambda_code_dir = os.path.join(current_dir, "..")  # Assuming the Lambda
code is in the parent directory
```

```
    # Run Bandit analysis
    bandit_issues = run_bandit_analysis(lambda_code_dir)

    # Run custom static checks
    custom_issues = run_custom_static_checks(os.path.join(lambda_code_dir,
"test_function.py"))

    # Combine all issues
    all_issues = bandit_issues + custom_issues
    return all_issues
```

This a static_analysis.py which is used to run static code analysis on any application.

After we create this library we test this library by creating python file called as test_function.py where we run all types of operations which mainly focuses on operation which might cause a security threat.

```
import os
import hashlib
import requests
from my_lambda_security_lib.lambda_wrapper import secure_lambda,
start_dynamic_monitoring, stop_dynamic_monitoring

@secure_lambda
def test_function(event, context):
    print("Test function has started.")

    # 1. Environment variable access
    secret_key = os.getenv("SECRET_KEY", "default_secret")
    print(f"Secret Key: {secret_key}")

    # 2. File operation: Write to a file
    with open("test_file.txt", "w") as file:
        file.write("This is a test file.")

    # 3. File operation: Read from a file
    with open("test_file.txt", "r") as file:
        content = file.read()
    print(f"File Content: {content}")

    # 4. Hashing operation (security-sensitive)
    # Introducing a weak cryptographic algorithm (MD5)
    password = "SuperSecretPassword123"
    hashed_password = hashlib.md5(password.encode()).hexdigest()  # This
should be flagged
    print(f"Hashed Password (MD5): {hashed_password}")

    # 5. Introducing the use of eval() which is a security risk
```

```python
    user_input = "1 + 2"
    result = eval(user_input)  # This should be flagged
    print(f"Eval result: {result}")

    # 6. Network operation: HTTP GET request
    response = requests.get("https://jsonplaceholder.typicode.com/todos/1")
    print(f"HTTP GET Response: {response.status_code}, Content:
{response.json()}")

    # 7. Exception Handling
    try:
        result = 10 / 0  # This will raise an exception
    except ZeroDivisionError as e:
        print(f"Caught an exception: {e}")

    # Return a result
    return {
        "statusCode": 200,
        "body": "Test function executed successfully."
    }

if __name__ == "__main__":
    # Simulate a Lambda event and context
    event = {}
    context = {}
    # Call the test function
    output = test_function(event, context)
    print(f"Function Output: {output}")
```

In this code we have imported lambda_wrapper from our library and totally tested mainly 7 operation which has to be taken care for the security of the application.

2.2 Testing the library

To test the library, we need to open terminal and execute the below line:

```
PS C:\Users\91903\OneDrive\Documents\libraryAndTestCode> python test_function.py
```

Once we the execute the function, static analysis code starts and shows if there are any issues in the code like shown below:

```
Static Analysis Issues Detected: ['Use of weak cryptography detected (MD5/SHA1). Consider using SHA256 or higher.']
```

Once the static code analysis is finished, dynamic analysis starts

```
Function call: get, Current depth: 3
CPU Usage: 0.00%, Memory Usage: 0.44%
Function call: __enter__, Current depth: 4
CPU Usage: 0.00%, Memory Usage: 0.44%
Function return: __enter__, Time taken: 0.000734 seconds
CPU Usage: 0.00%, Memory Usage: 0.44%
Function call: _qsize, Current depth: 4
CPU Usage: 0.00%, Memory Usage: 0.44%
Function return: _qsize, Time taken: 0.000891 seconds
Function call: _get, Current depth: 4
```

```
--- Detailed Summary ---
Total Execution Time: 4.460390 seconds

Function Execution Times:
- test_function: 4.371315 seconds
- getenv: 0.006247 seconds
- get: 0.002727 seconds
- __getitem__: 0.000289 seconds
- encodekey: 0.001969 seconds
- check_str: 0.000868 seconds
- patched_open: 0.004026 seconds
- __init__: 0.000610 seconds
- encode: 0.000763 seconds
- decode: 0.002554 seconds
- <module>: 1259081.652632 seconds
- patched_get: 4.337516 seconds
- request: 1259084.956600 seconds
- default_headers: 0.014501 seconds
- default_user_agent: 0.000279 seconds
- update: 0.087077 seconds
- __instancecheck__: 0.003171 seconds
- __subclasscheck__: 0.001631 seconds
- __subclasshook__: 0.000285 seconds
- __setitem__: 0.000386 seconds
- default_hooks: 0.001079 seconds
- <dictcomp>: 0.000354 seconds
- cookiejar_from_dict: 0.007905 seconds
- RLock: 0.000859 seconds
- __iter__: 0.000320 seconds
- <listcomp>: 0.009547 seconds
- deepvalues: 0.000243 seconds
- init_poolmanager: 0.005441 seconds
```

```
Maximum Function Call Depth: 27

Average CPU Usage: 0.37%
Peak CPU Usage: 104.20%
Average Memory Usage: 0.41%
Peak Memory Usage: 0.44%

Network Requests:
- URL: https://jsonplaceholder.typicode.com/todos/1, Method: GET, Status: 200, Data Sent: 0 bytes, Data Received: 83 bytes
Total Data Sent: 0 bytes
Total Data Received: 83 bytes

File Operations:
- File: test_file.txt, Mode: r
- File: test_file.txt, Mode: r

--- End of Summary ---
Function Output: {'statusCode': 200, 'body': 'Test function executed successfully.'}
```

3 Deploying to AWS

Once it is locally developed and tested, it is now that deploy and integrate our library into AWS Lambda

To do that , we ned to deploy our custom pckage and all its dependencies in the same envirnoment hence we containerized our project using docker.

```
container_name=lambda_docker
docker_image=aws_lambda_builder_image
docker run -td --name=$container_name $docker_image
docker cp ./requirements.txt $container_name:/

docker exec -i $container_name /bin/bash < ./docker_install.sh
docker cp $container_name:/python.zip python.zip
docker stop $container_name
docker rm $container_name
```

An envirnoment was described using a Dockerfile.

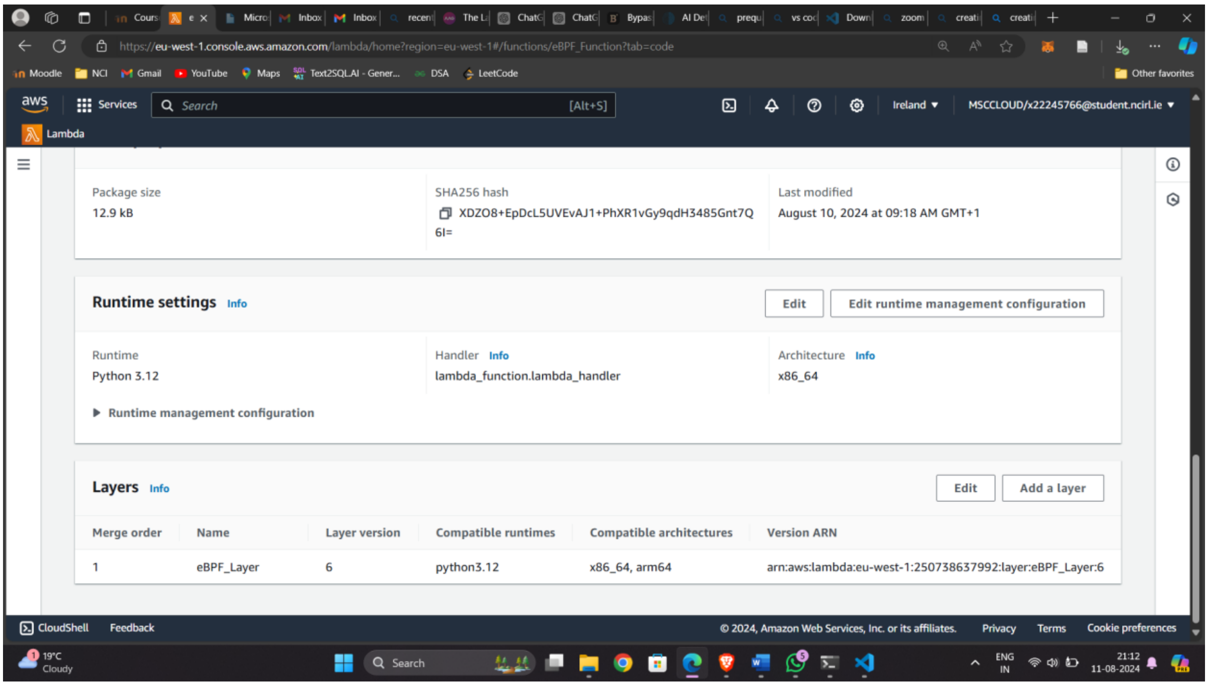
```
virtualenv --python=/user/bin/python3/ python
source python/bin/activate
pip install -r requirements.txt -t python/lib/python3/site-packages

zip -r9 python.zip python
```

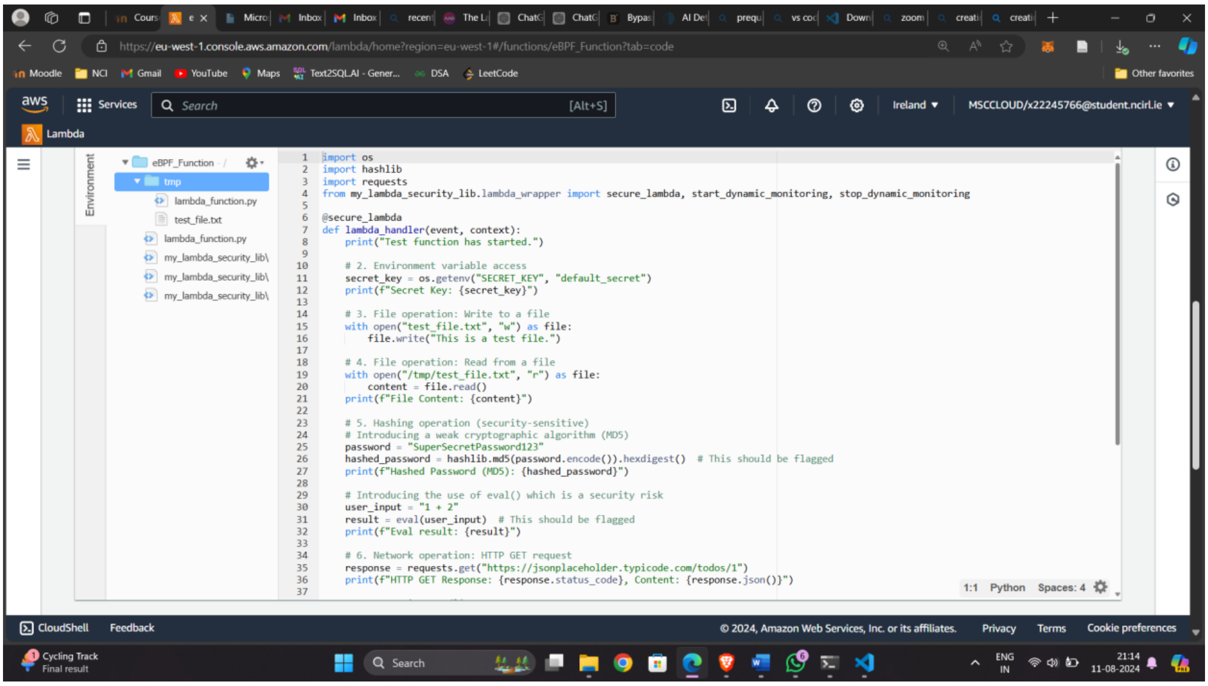
Which creates an Docker imagem

```
REPOSITORY                 TAG       IMAGE ID       CREATED       SIZE
aws_lambda_builder_image   latest    63b2685909f8   2 days ago    312MB
```

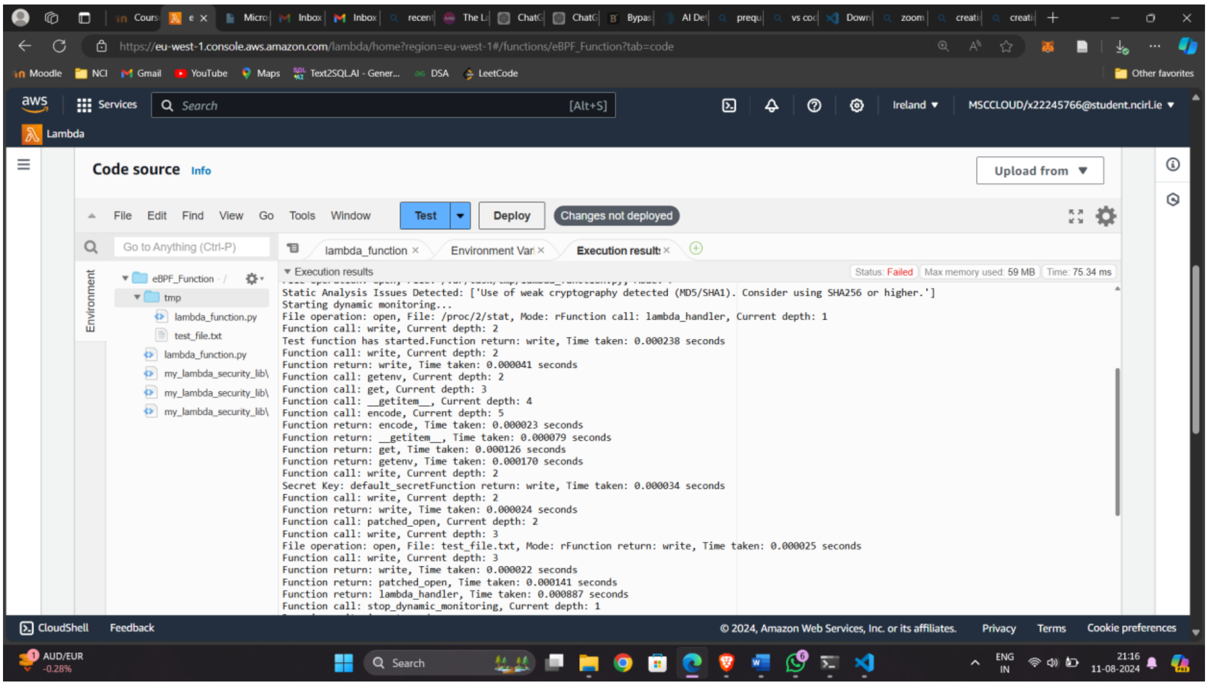
Then from the dockerimage we exract the file and create a lambda layer in AWS Lambda.



Lambda
Package size
12.9 kB
SHA256 hash
XDZO8+EpDcL5UVEvAJ1+PhXR1vGy9qdH3485Gnt7Q6I=
Last modified
August 10, 2024 at 09:18 AM GMT+1
Runtime settings Info
Edit
Edit runtime management configuration
Runtime
Python 3.12
Handler Info
lambda_function.lambda_handler
Architecture Info
x86_64
Runtime management configuration
Layers Info
Edit
Add a layer
Merge order
Name
Layer version
Compatible runtimes
Compatible architectures
Version ARN
1
eBPF_Layer
6
python3.12
x86_64, arm64
arn:aws:lambda:eu-west-1:250738637992:layer:eBPF_Layer:6
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



Lambda
eBPF_Function
tmp
lambda_function.py
test_file.txt
lambda_function.py
import os
import hashlib
import requests
from my_lambda_security_lib.lambda_wrapper import secure_lambda, start_dynamic_monitoring, stop_dynamic_monitoring
@secure_lambda
def lambda_handler(event, context):
print("Test function has started.")
# 2. Environment variable access
secret_key = os.getenv("SECRET_KEY", "default_secret")
print(f"Secret Key: {secret_key}")
# 3. File operation: Write to a file
with open("test_file.txt", "w") as file:
file.write("This is a test file.")
# 4. File operation: Read from a file
with open("/tmp/test_file.txt", "r") as file:
content = file.read()
print(f"File Content: {content}")
# 5. Hashing operation (security-sensitive)
# Introducing a weak cryptographic algorithm (MD5)
password = "SuperSecretPassword123"
hashed_password = hashlib.md5(password.encode()).hexdigest() # This should be flagged
print(f"Hashed Password (MD5): {hashed_password}")
# Introducing the use of eval() which is a security risk
user_input = "1 + 2"
result = eval(user_input) # This should be flagged
print(f"Eval result: {result}")
# 6. Network operation: HTTP GET request
response = requests.get("https://jsonplaceholder.typicode.com/todos/1")
print(f"HTTP GET Response: {response.status_code}, Content: {response.json()}")
1:1 Python Spaces: 4
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



References

Microsoft (2021) *Download visual studio code - MAC, linux, windows, RSS*. Available at: https://code.visualstudio.com/Download (Accessed: 12 August 2024).

Download python (no date) *Python.org*. Available at: https://www.python.org/downloads/ (Accessed: 12 August 2024).

Introduction, tutorials & community resources (no date) *eBPF*. Available at: https://ebpf.io/ (Accessed: 12 August 2024).

Bastien (2023) *How to create your own library in Python, The Python You Need*. Available at: https://thepythonyouneed.com/how-to-create-your-own-library-in-python (Accessed: 12 August 2024).

Accelerated Container Application Development (2024) *Docker*. Available at: https://www.docker.com/ (Accessed: 12 August 2024).

Aws Lambda: The ultimate guide (no date) *AWS Lambda: The Ultimate Guide*. Available at: https://www.serverless.com/aws-lambda (Accessed: 12 August 2024).