

Project Solution



| | |
|--|------------------------------|
| Task 1: Hardening The Service | 2 |
| Task Challenge: | Error! Bookmark not defined. |
| Task Solution: | Error! Bookmark not defined. |
| Task 2: Planning Layer Of Defense | Error! Bookmark not defined. |
| Task Challenge: | 3 |
| Task Solution: | Error! Bookmark not defined. |
| Task 3: Integrating the Yara Scanning function With S3 Bucket | 3 |
| Task Challenge: | 4 |
| Task Solution: | Error! Bookmark not defined. |
| Task 4: Scanning Suspicious Files | 4 |
| Task Challenge: | 5 |
| Task solution: | Error! Bookmark not defined. |
| Task 5: Defense-in-Depth | 5 |
| Task Challenge: | 6 |
| Task Solution: | Error! Bookmark not defined. |
| Task 6: Complete Architecture Diagram | 6 |
| Project Challenge: | 7 |
| Project Solution: | Error! Bookmark not defined. |

For this project, an AWS command line, as well as console access, is required.
Set up the command line access using the instructions listed below and based on your machine's operating system.

Task 1: Hardening The Service

Create and apply a bucket policy using the command line or the AWS UI and apply it on the file-scan-upload bucket to make it defensible against binary file uploads.

Using what you learned from the course, harden the bucket so that anyone can upload only the suitable types of files using AWS Policies.

Submit the policy as a JSON file and name it bucket_restriction.json.

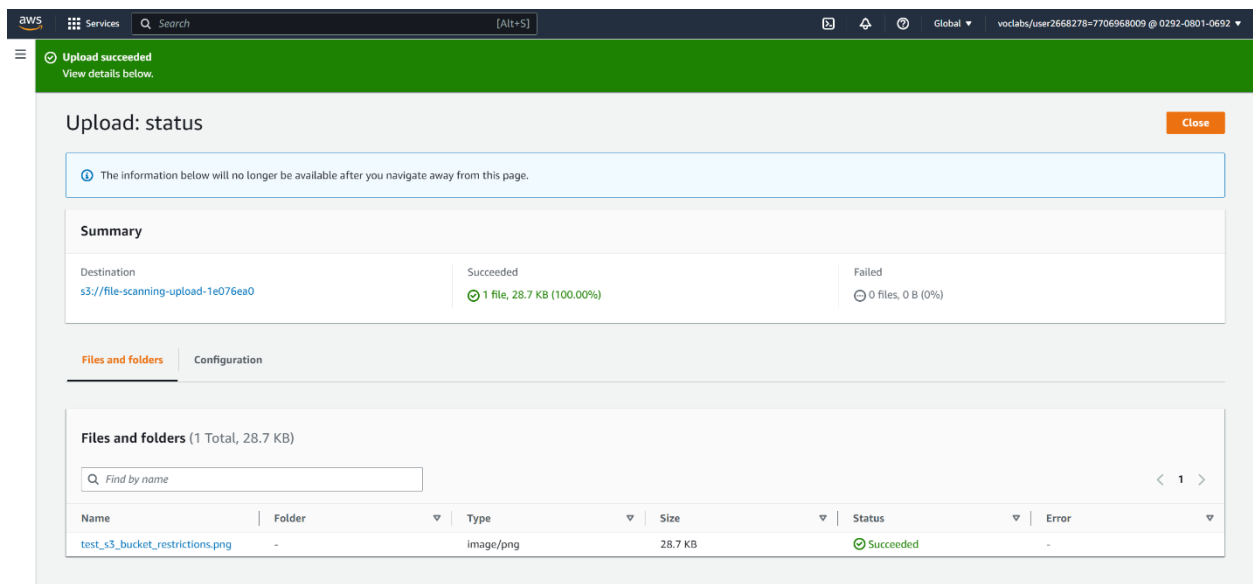
Also, submit a snapshot below or as a jpeg of the command line or AWS UI showing that an error pops up when uploading a non-image filetype. **Name the snapshot as failed_upload.jpeg if you include it as a separate file.**

Please include an answer to the question below.

What did you find when attempting to upload your files? How might you harden this bucket, and why is that the best solution?

- ➔ To add a restriction to the s3 bucket, in the bucket_restrction.json, I first add a permission to allow users to upload any files with extensions of **.png, jpg** and **.gif**. Once the bucket has allowed the image types, I proceed to implement another permission to block all the files that are not of image types (i.e not ending with **.png, jpg** and **.gif**). This will complete the **bucket_restrction.json** file, which is a policy attached to the s3 bucket. As a result, I could successfully upload images (successful_upload.png) and prevent non-images uploads (failed_upload.png).

successful_upload.png



The screenshot shows the AWS Management Console interface. At the top, there's a green banner indicating "Upload succeeded" with a link to "View details below." Below this, the "Upload: status" window is open, showing a summary of the upload. The summary indicates that the file was successfully uploaded to the destination s3://file-scanning-upload-1e076ea0. The "Files and folders" section shows a table with one file: "test_s3_bucket_restrictions.png", which is an image/png file, 28.7 KB in size, and has a status of "Succeeded".

| Name | Folder | Type | Size | Status | Error |
|---------------------------------|--------|-----------|---------|-----------|-------|
| test_s3_bucket_restrictions.png | - | image/png | 28.7 KB | Succeeded | - |

failed_upload.png

Upload failed
View details below.

Upload: status Close

The information below will no longer be available after you navigate away from this page.

Summary

| | | |
|---|--------------------------------|------------------------------------|
| Destination s3://file-scanning-upload-1e076ea0 | Succeeded 0 files, 0 B (0%) | Failed 1 file, 16.0 B (100.00%) |
|---|--------------------------------|------------------------------------|

Files and folders | Configuration

Files and folders (1 Total, 16.0 B)

Find by name

| Name | Folder | Type | Size | Status | Error |
|---------------------------------|--------|------------|--------|--------|---------------|
| test_s3_bucket_restrictions.txt | - | text/plain | 16.0 B | Failed | Access Denied |

Task 2: Planning Layer Of Defense

Task Challenge:

What AWS service would you propose to the WPO team to build the Yara scanning feature on the S3 uploads and why?

- ➔ I would suggest using Lambda function as a service to integrate Yara scanning into the applications. The reason is being we want to minimize the cost of AWS service. Cost only incurs when the Lambda function triggers, which is when an image is uploaded.

Task 3: Integrating the Yara Scanning function With S3 Bucket

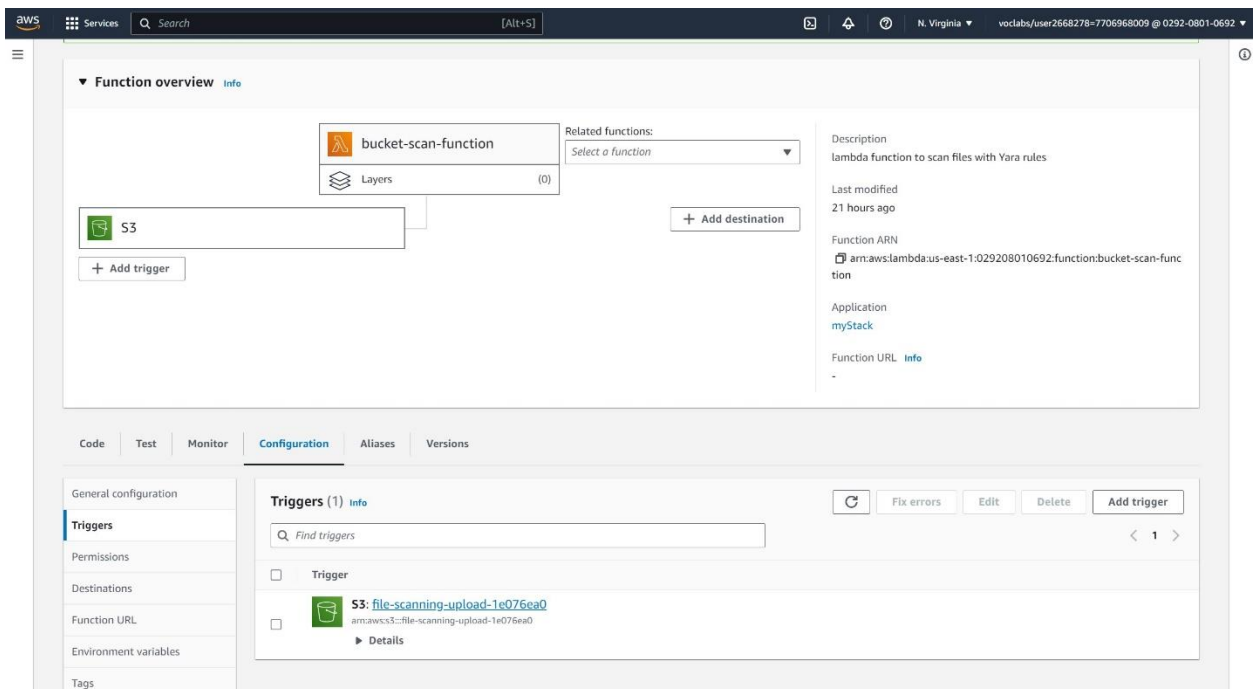
Task Challenge:

Launch the AWS dashboard and navigate to the Lambda functions page. You will notice a lambda called **bucket-scan-function**.

Address the following in the template:

- What changes should you make to the lambda function's settings so that it can start scanning the file uploads?
 - Perform the change you have suggested above and submit a snapshot of the change you have made and name the file as **lambda_s3_trigger.jpg**.
- ➔ What I would do is to add a trigger event for this **bucket-scan-function** so that every time a new image is uploaded, the functions will scan the file.

lambda_s3_trigger.jpg



Task 4: Scanning Suspicious Files

Task Challenge:

Upload the provided three images to the **file-scan-upload-[random]** bucket and answer the following:

- Using the cloudwatch logs, determine and provide the **detection_name** that is triggered within the image files.
- Based on the logs, the lambda function writes the result of the scan to another AWS service. Which service is it? Navigate to the service and post the JSON result of the suspicious file saved in that service.

➔ The detection name is: ***Udacity_malware_alert***

The screenshot displays the AWS CloudWatch Logs console. The left sidebar shows the navigation menu with 'Logs' selected. The main panel shows the logs for a Lambda function. The logs are as follows:

| Time | Message |
|-------------------------------|--|
| 2023-08-23T13:07:57.926-07:00 | END RequestId: 800b11c8-bc76-44af-9989-c567ffafae0a |
| 2023-08-23T13:07:57.926-07:00 | REPORT RequestId: 800b11c8-bc76-44af-9989-c567ffafae0a Duration: 673.13 ms Billed Duration: 674 ms Memory Size: 128 MB Max Memory Used: 104 MB |
| 2023-08-23T13:12:25.063-07:00 | START RequestId: 6134dbfb-9055-4d16-ab41-3b202fd02d87 Version: \$LATEST |
| 2023-08-23T13:12:25.064-07:00 | Script starting at 2023/08/23 20:12:25 UTC |
| 2023-08-23T13:12:25.065-07:00 | creating local path for file %s /tmp/file-scanning-upload-1e076ea0/meter-1.jpg |
| 2023-08-23T13:12:25.344-07:00 | file scanning to begin |
| 2023-08-23T13:12:25.344-07:00 | /var/task |
| 2023-08-23T13:12:25.344-07:00 | /var/task/yara-rules |
| 2023-08-23T13:12:25.685-07:00 | {'scan_performed': 'No', 'scan_result': 'Not-detected', 'detection_rule': 'N/A'} |
| 2023-08-23T13:12:25.686-07:00 | {'scan_performed': 'No', 'scan_result': 'Not-detected', 'detection_rule': 'N/A'} |
| 2023-08-23T13:12:25.686-07:00 | sending details to SQS Queue |
| 2023-08-23T13:12:25.985-07:00 | END RequestId: 6134dbfb-9055-4d16-ab41-3b202fd02d87 |
| 2023-08-23T13:12:25.985-07:00 | REPORT RequestId: 6134dbfb-9055-4d16-ab41-3b202fd02d87 Duration: 921.66 ms Billed Duration: 922 ms Memory Size: 128 MB Max Memory Used: 105 MB |
| 2023-08-23T13:12:26.475-07:00 | START RequestId: e7a9b174-fc38-4da2-8ac6-9f40476a3838 Version: \$LATEST |
| 2023-08-23T13:12:26.476-07:00 | Script starting at 2023/08/23 20:12:26 UTC |
| 2023-08-23T13:12:26.477-07:00 | creating local path for file %s /tmp/file-scanning-upload-1e076ea0/meter-3.jpg |
| 2023-08-23T13:12:26.604-07:00 | file scanning to begin |
| 2023-08-23T13:12:26.604-07:00 | /var/task |
| 2023-08-23T13:12:26.604-07:00 | /var/task/yara-rules |
| 2023-08-23T13:12:26.905-07:00 | {'scan_performed': 'Yes', 'scan_result': 'Detected', 'detection_rule': ['Udacity_malware_alert']} |
| 2023-08-23T13:12:26.906-07:00 | {'scan_performed': 'Yes', 'scan_result': 'Detected', 'detection_rule': ['Udacity_malware_alert']} |
| 2023-08-23T13:12:26.906-07:00 | sending details to SQS Queue |
| 2023-08-23T13:12:27.245-07:00 | END RequestId: e7a9b174-fc38-4da2-8ac6-9f40476a3838 |
| 2023-08-23T13:12:27.245-07:00 | REPORT RequestId: e7a9b174-fc38-4da2-8ac6-9f40476a3838 Duration: 769.87 ms Billed Duration: 770 ms Memory Size: 128 MB Max Memory Used: 105 MB |

No newer events at this moment. Auto retry paused. [Resume](#)

[Back to top](#)

- ➔ Based on the logs, the lambda function writes the result of the scan to AWS SQS Queue service. The result of the suspicious file is saved in JSON file: **suspicious_file.json**

```
{  
  "bucket": "file-scanning-upload-1e076ea0",  
  "key": "meter-3.jpg",  
  "yara-scan": {  
    "scan_performed": "Yes",  
    "scan_result": "Detected",  
    "detection_rule": [  
      "Udacity_malware_alert"  
    ]  
  }  
}
```

Task 5: Defense-in-Depth

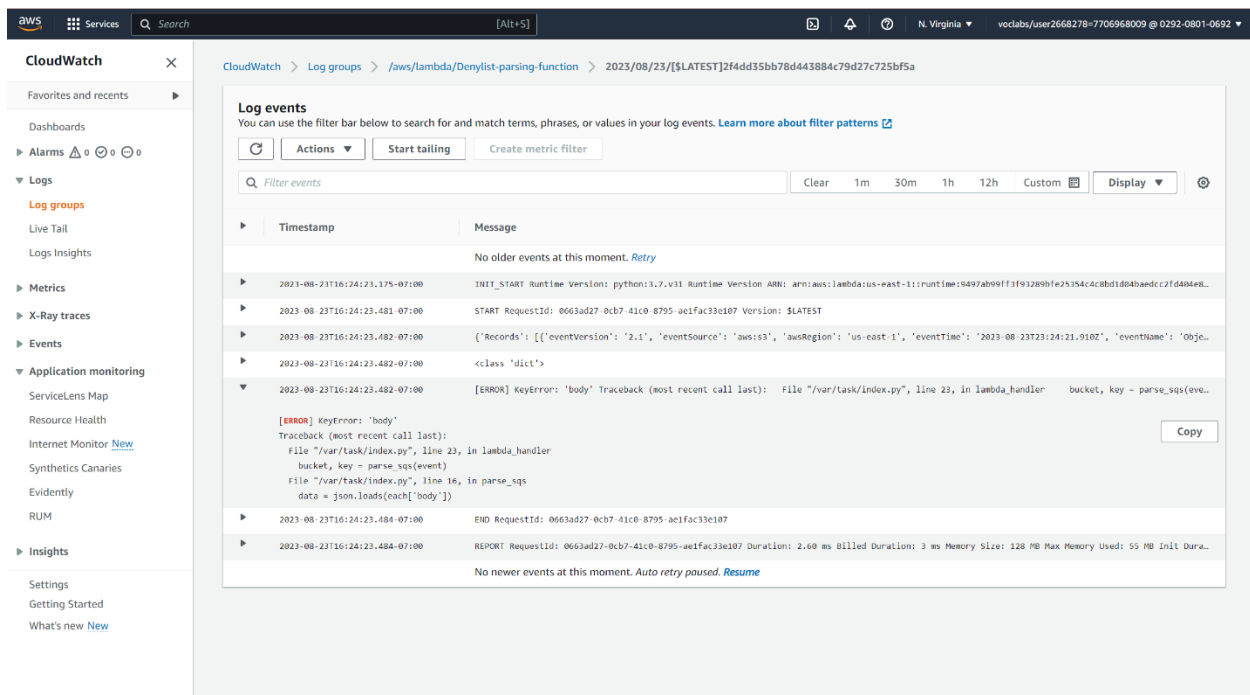
Task Challenge:

Task goal is to modify the Denylist-parsing-function lambda code so that it is able to match the uploaded files against a set of pre-defined hashes.

Submit the modified lambda function's code either as "**denylist_match.py**" or as a screenshot in the template as evidence. The snapshot should show the changes you have made in the code.

Also, submit a snapshot of the cloudwatch logs showing the successful hash match for the same suspicious image file which triggered the Yara alert. Name the file as **denylist_cloudwatch.jpg**.

- ➔ After modifying the Lambda function, I reuploaded the images. However, there were some errors in the functions that prevents the Lambda from yielding expected results.



Task 6: Complete Architecture Diagram

Project Challenge:

Modify the starter architecture diagram and add it to the template showing evidence of the following elements (you could also include the image file in your submission):

- The flow of file from the User to the file-upload S3 bucket
- Bucket-scan-function Lambda Trigger to scan files through the Yara engine.
- SQS queue message invoking the Denylist-parsing-function.
- Presence of Hashlist in a separate S3 bucket, which the Denylist-parsing-function will read.

