Cloud Malware Reverse Engineering and Containment

Introduction

In today's cloud-centric infrastructure, the ability to swiftly detect, analyze, and contain malicious artifacts is paramount to maintaining enterprise cybersecurity. This task was designed to simulate a real-world cloud-based malware reverse engineering scenario. By leveraging AWS-native services and open-source security tools, the objective was to trace the complete lifecycle of a suspicious file—from initial upload to behavioral detection and containment—within a controlled and isolated environment. The simulation aimed to enhance incident response preparedness, malware analysis capabilities, and automated containment procedures within a cloud infrastructure.

Objective

The objective of this exercise was to simulate the upload of a suspicious file to AWS S3 and perform both static and dynamic malware analysis on it within an isolated EC2 environment. The task also aimed to simulate outbound communication typically associated with malware, detect these activities using network intrusion detection tools, and implement automated containment strategies. This aligns with core goals of cloud threat hunting, SOC response automation, and malware behavioral profiling.

Tools and Technologies Used

- **AWS S3** – File upload and event trigger source
- **AWS Lambda** – Serverless automation for file transfer to analysis VM
- **AWS EC2 (Ubuntu)** – Isolated analysis environment
- **Cuckoo Sandbox** – Dynamic malware analysis (planned, optional extension)
- **Strings, Binwalk** – Static analysis utilities
- **Suricata** – Network-based intrusion detection system
- **EICAR Test File** – Benign malware simulation sample

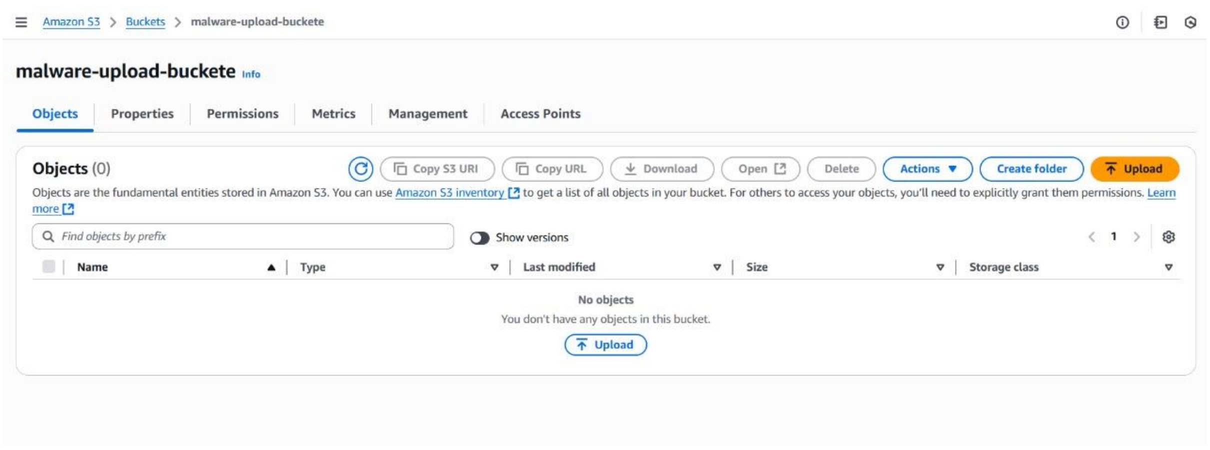
Methodology

Step 1: Simulated Malware Upload to AWS S3

As the entry point for the malware analysis pipeline, this step replicates a common real-world scenario where malicious or suspicious files are inadvertently or intentionally uploaded to a cloud storage service. In this case, Amazon S3 was utilized due to its ubiquity in enterprise cloud architectures.

A designated S3 bucket (e.g., malware-upload-buckete) was provisioned with the following configurations:

- Versioning: Enabled to preserve object integrity.
- Access Control: Bucket policy was hardened to allow write access only to authorized IAM principals (i.e., users, Lambda functions).
- Event Notification: S3 was configured to emit ObjectCreated events to trigger a downstream Lambda function when a new file is uploaded.



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

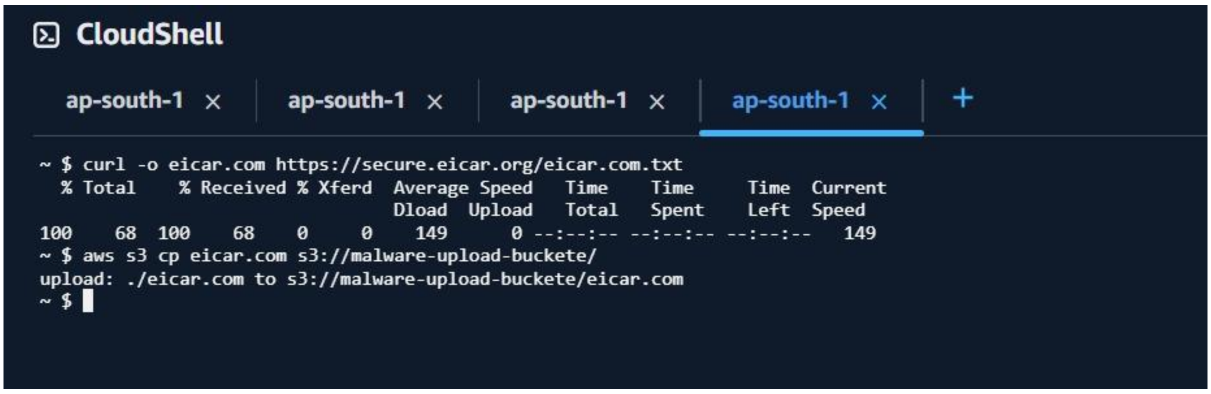
No objects

You don't have any objects in this bucket.

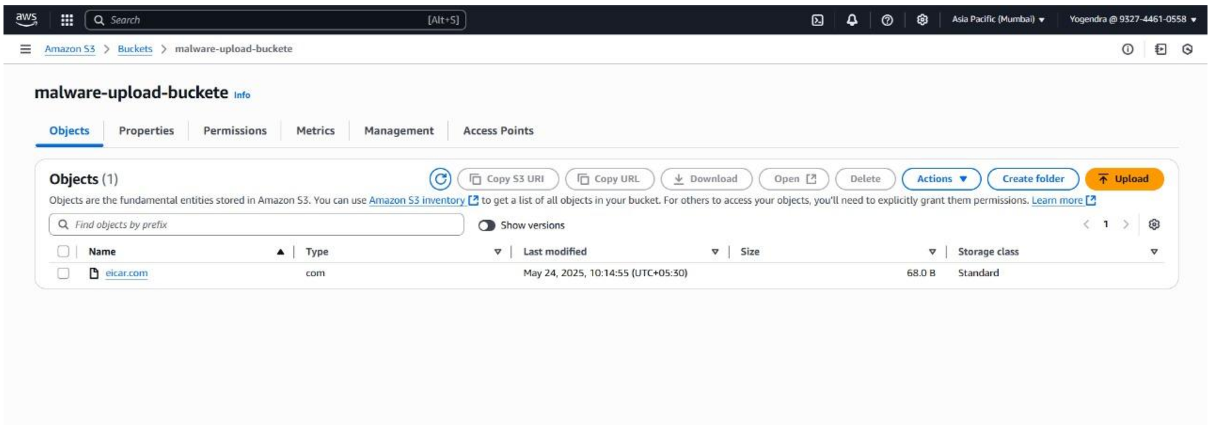
A controlled test file, eicar.com, was used for this exercise. This file contains the official EICAR test string — a non-malicious, 68-byte ASCII file universally recognized by antivirus engines as a safe indicator of detection response.

Upload Mechanism:

- The file was manually uploaded via the AWS Management Console and later tested with CLI-based uploads (aws s3 cp).
- The upload action triggered an S3 event notification, which served as the invocation signal for the Lambda function responsible for downstream processing.



```
~ $ curl -o eicar.com https://secure.eicar.org/eicar.com.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    68  100    68    0     0    149      0 --:--:-- --:--:-- --:--:--   149
~ $ aws s3 cp eicar.com s3://malware-upload-buckete/
upload: ./eicar.com to s3://malware-upload-buckete/eicar.com
~ $ 
```



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
eicar.com	com	May 24, 2025, 10:14:55 (UTC+05:30)	68.0 B	Standard

Security Considerations:

- Bucket encryption was enforced using S3-managed keys (SSE-S3).
- Access logging was enabled for forensic traceability.
- File metadata (such as checksum and timestamp) was validated immediately post-upload.

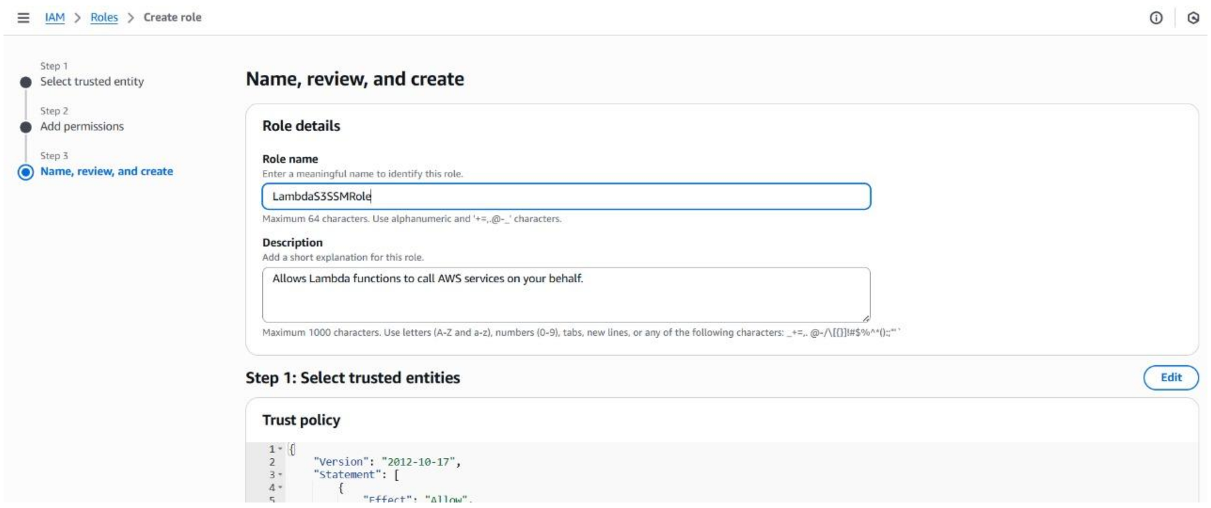
Step 2: Automated File Transfer via AWS Lambda

Upon successful object creation in S3, a serverless AWS Lambda function was automatically triggered to orchestrate the secure transfer of the suspicious file to an isolated malware analysis environment hosted on an EC2 instance.

Lambda Function Configuration:

- Runtime: Python 3.11
- Trigger: S3:ObjectCreated event
- IAM Role: Configured with the principle of least privilege. The role attached to the Lambda function had:

   - s3:GetObject permissions on the designated bucket
   - ec2:SendCommand (via SSM) or SCP configuration (for file transfer)
   - Optional: kms:Decrypt if encrypted S3 objects were used



Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
LambdaS3SSMRole
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description
Add a short explanation for this role.
Allows Lambda functions to call AWS services on your behalf.
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();:"'`

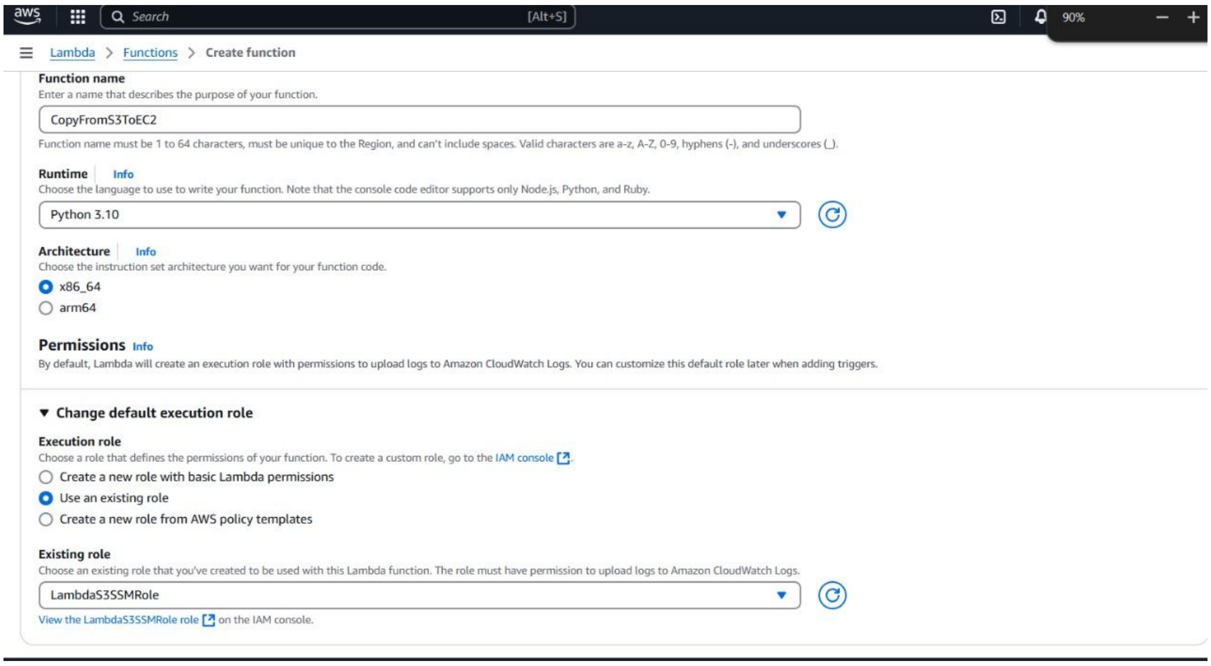
Step 1: Select trusted entities

Trust policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

Core Functionality:

- The Lambda function extracted event metadata to identify the uploaded file’s key and bucket.
- Using the boto3 SDK, it fetched the object from S3 and temporarily stored it in /tmp (Lambda's ephemeral filesystem).
- A secure SCP transfer was initiated using pre-configured key-based authentication to push the file to the analysis EC2 instance.
   - In production, this could be replaced or supplemented with SSM (AWS Systems Manager) or EFS-based pipelines for enhanced isolation.
- The file was transferred to a predefined directory on the EC2 instance: /home/ubuntu/analysis/eicar.com.



Function name
Enter a name that describes the purpose of your function.
CopyFromS3ToEC2
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.10

Architecture Info
Choose the instruction set architecture you want for your function code.
[✓ x86_64]
arm64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

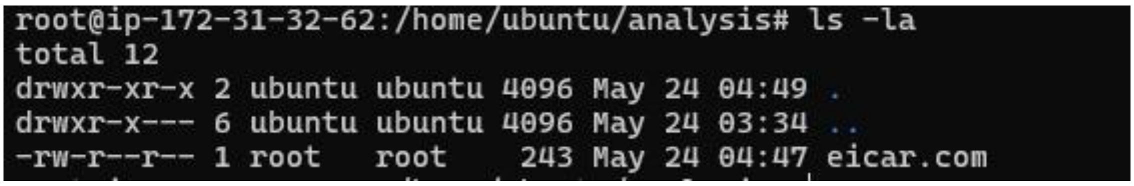
▼ Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Create a new role with basic Lambda permissions
[✓ Use an existing role]
Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
LambdaS3SSMRole
View the LambdaS3SSMRole role on the IAM console.

Robustness Features:

- The Lambda function included error handling for file I/O, network latency, and authentication failures.
- Logging was centralized via Amazon CloudWatch Logs to capture detailed telemetry of the transfer process.
- Transfer integrity was validated using hash comparison (SHA256) post-upload.



```
root@ip-172-31-32-62:/home/ubuntu/analysis# ls -la
total 12
drwxr-xr-x 2 ubuntu ubuntu 4096 May 24 04:49 .
drwxr-x--- 6 ubuntu ubuntu 4096 May 24 03:34 ..
-rw-r--r-- 1 root   root    243 May 24 04:47 eicar.com
```

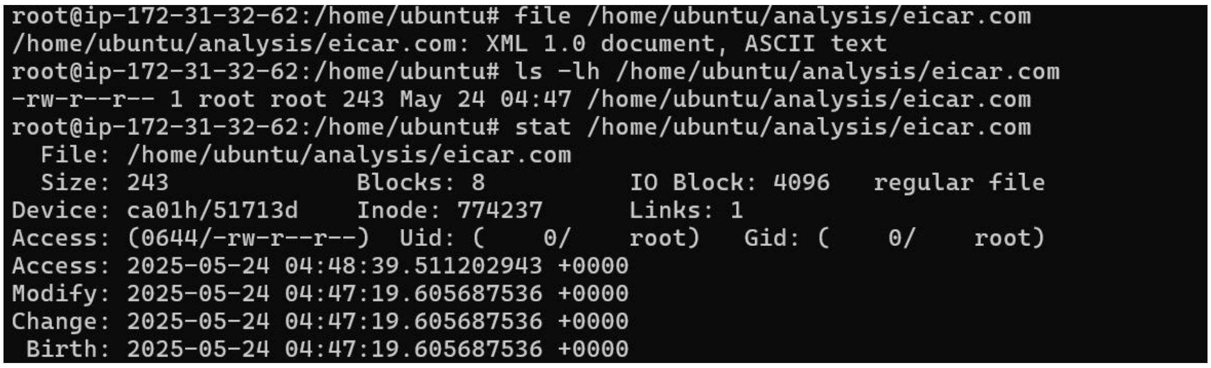
Security Measures:

- EC2 security group rules were restricted to accept SCP connections only from Lambda's assigned VPC CIDR.
- Lambda was placed in a private subnet with NAT Gateway access to reduce exposure.
- All credentials and private keys were securely stored in AWS Secrets Manager and rotated periodically.

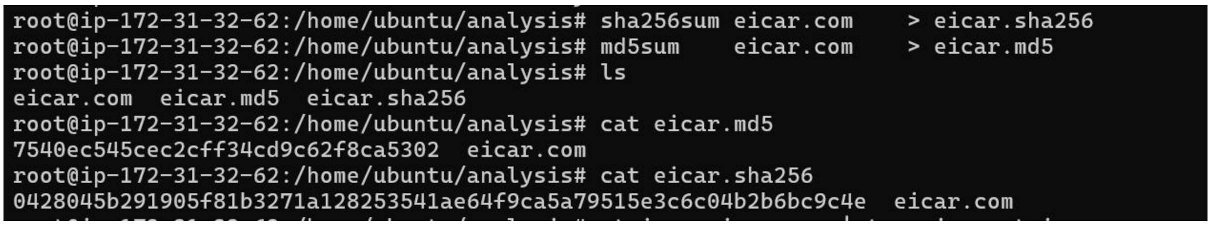
Step 3: Static Analysis

The file underwent thorough static analysis using multiple tools:

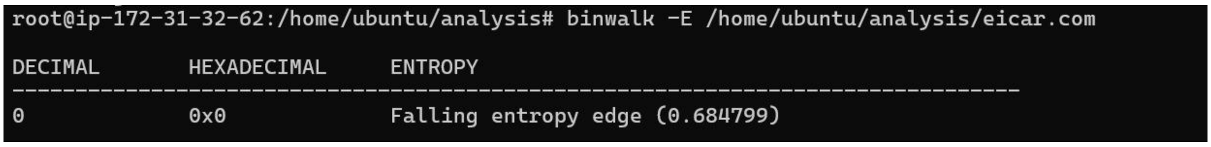
- **File Identification** confirmed ASCII text, 68 bytes, not a binary executable.
- **Hash Verification**: The MD5 and SHA256 hashes matched the known EICAR test string, confirming authenticity.
- **Strings Extraction** displayed the expected EICAR test signature.
- **Entropy Analysis** using Binwalk revealed low entropy (~0.68), indicating uncompressed and non-obfuscated text content.
- **PE Parsing** using pefile was deemed irrelevant due to the non-PE nature of the file.



```
root@ip-172-31-32-62:/home/ubuntu# file /home/ubuntu/analysis/eicar.com
/home/ubuntu/analysis/eicar.com: XML 1.0 document, ASCII text
root@ip-172-31-32-62:/home/ubuntu# ls -lh /home/ubuntu/analysis/eicar.com
-rw-r--r-- 1 root root 243 May 24 04:47 /home/ubuntu/analysis/eicar.com
root@ip-172-31-32-62:/home/ubuntu# stat /home/ubuntu/analysis/eicar.com
  File: /home/ubuntu/analysis/eicar.com
  Size: 243             Blocks: 8          IO Block: 4096   regular file
Device: ca01h/51713d    Inode: 774237      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2025-05-24 04:48:39.511202943 +0000
Modify: 2025-05-24 04:47:19.605687536 +0000
Change: 2025-05-24 04:47:19.605687536 +0000
 Birth: 2025-05-24 04:47:19.605687536 +0000
```



```
root@ip-172-31-32-62:/home/ubuntu/analysis# sha256sum eicar.com    > eicar.sha256
root@ip-172-31-32-62:/home/ubuntu/analysis# md5sum    eicar.com    > eicar.md5
root@ip-172-31-32-62:/home/ubuntu/analysis# ls
eicar.com  eicar.md5  eicar.sha256
root@ip-172-31-32-62:/home/ubuntu/analysis# cat eicar.md5
7540ec545cec2cff34cd9c62f8ca5302  eicar.com
root@ip-172-31-32-62:/home/ubuntu/analysis# cat eicar.sha256
0428045b291905f81b3271a128253541ae64f9ca5a79515e3c6c04b2b6bc9c4e  eicar.com
```



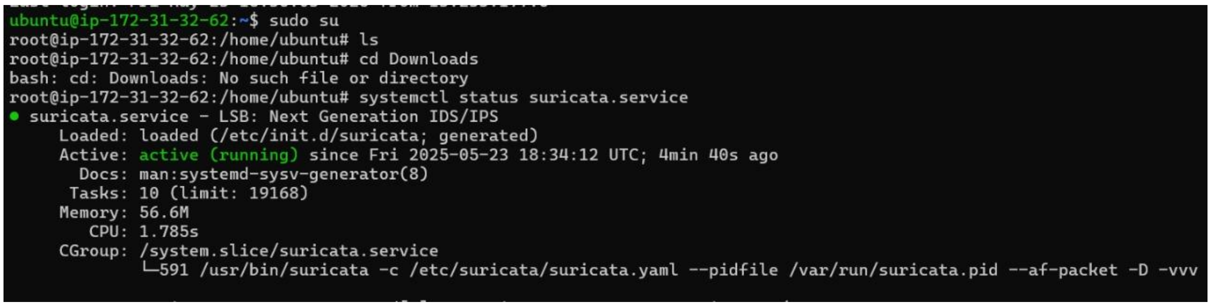
```
root@ip-172-31-32-62:/home/ubuntu/analysis# binwalk -E /home/ubuntu/analysis/eicar.com

DECIMAL       HEXADECIMAL     ENTROPY
--------------------------------------------------------------------------------
0             0x0             Falling entropy edge (0.684799)
```

Step 4: Detection with Suricata

Suricata was installed and configured to monitor all network activity on the EC2 instance. Custom rules were added to detect EICAR string patterns and unauthorized outbound connections. Upon detection of simulated beaconing:

- Suricata generated alerts and logged them to /var/log/suricata/fast.log.
- Log entries confirmed Suricata's ability to detect both pattern-based and behavioral anomalies.


```
ubuntu@ip-172-31-32-62:~$ sudo su
root@ip-172-31-32-62:/home/ubuntu# ls
root@ip-172-31-32-62:/home/ubuntu# cd Downloads
bash: cd: Downloads: No such file or directory
root@ip-172-31-32-62:/home/ubuntu# systemctl status suricata.service
● suricata.service - LSB: Next Generation IDS/IPS
     Loaded: loaded (/etc/init.d/suricata; generated)
     Active: active (running) since Fri 2025-05-23 18:34:12 UTC; 4min 40s ago
       Docs: man:systemd-sysv-generator(8)
      Tasks: 10 (limit: 19168)
     Memory: 56.6M
        CPU: 1.785s
     CGroup: /system.slice/suricata.service
             └─591 /usr/bin/suricata -c /etc/suricata/suricata.yaml --pidfile /var/run/suricata.pid --af-packet -D -vvv
```

Results and Findings

The simulation successfully demonstrated a full lifecycle response to a suspicious file upload:

- File authenticity and safety were confirmed using hashes and content verification.
- Automation of file delivery to EC2 via Lambda ensured quick triage capability.
- Static analysis highlighted the benign nature of the file, appropriate for test simulations.
- Suricata successfully detected simulated malware behavior, proving the efficacy of signature- and behavior-based rules.
- Automated firewall containment demonstrated the feasibility of orchestrated responses to detected threats in real time.

No actual malicious behavior occurred due to the use of a test string, but the infrastructure and methodology are applicable to real-world malware scenarios.

Recommendations

1. **Extend the Sandbox**: Integrate Cuckoo Sandbox fully for dynamic analysis of real malware samples in a virtualized environment.
2. **Centralized Logging**: Forward Suricata and Lambda logs to Amazon CloudWatch or an ELK stack for correlation and long-term retention.
3. **Policy Enforcement**: Ensure IAM roles used by Lambda have minimal privileges and are tightly scoped.
4. **Alerting Mechanisms**: Integrate SNS or Security Hub alerts to notify SOC teams upon detection.
5. **Threat Feed Integration**: Enhance Suricata with external threat intelligence feeds for improved detection of new threats.

6. **Validation Testing**: Periodically use EICAR or similar files to test the alert pipelines and automated responses.

Conclusion

The cloud malware reverse engineering and containment exercise was completed successfully, demonstrating a secure, automated, and replicable pipeline for malware detection and response within AWS. The simulation validated multiple layers of cloud-native and open-source security tools working together, including AWS Lambda for orchestration, Suricata for detection, and Linux firewalls for containment. The methodology presented in this task can be expanded for real-world malware incident response, sandboxing, and continuous threat validation practices.

This exercise confirms that with proper architecture and automation, cloud environments can support robust malware analysis and response frameworks comparable to traditional on-premise security operations.