PRJ-SEC-012: Automated Incident Response & Forensics

Author: Mo Suleiman | **Project:** PRJ-SEC-012 | **Certification:** AWS Certified Security – Specialty

1. Project Overview

This project implements an automated workflow for **digital forensics and incident response (DFIR)** in the AWS cloud. When a security incident occurs, a rapid and consistent response is crucial to contain the threat and preserve evidence for investigation. Manual response is often too slow and can lead to mistakes that corrupt evidence. This project automates the initial, critical steps of evidence acquisition.

Upon detection of a high-severity **Amazon GuardDuty** finding on an EC2 instance, an **AWS Step Functions** workflow is triggered via **Amazon EventBridge**. This workflow orchestrates the entire evidence collection process: it isolates the compromised instance, takes a snapshot of its EBS volume (disk forensics), and captures a memory dump (memory forensics) using AWS Systems Manager. The collected evidence is stored securely in a dedicated S3 bucket, ready for a forensic analyst to investigate in a safe, isolated environment.

To make this project as realistic as possible, we use the **Official AWS GuardDuty Tester** to simulate real malicious behavior (like cryptocurrency mining and C2 communication) on an actual EC2 instance, triggering genuine GuardDuty alerts and executing the full end-to-end workflow.

GRC Mapping

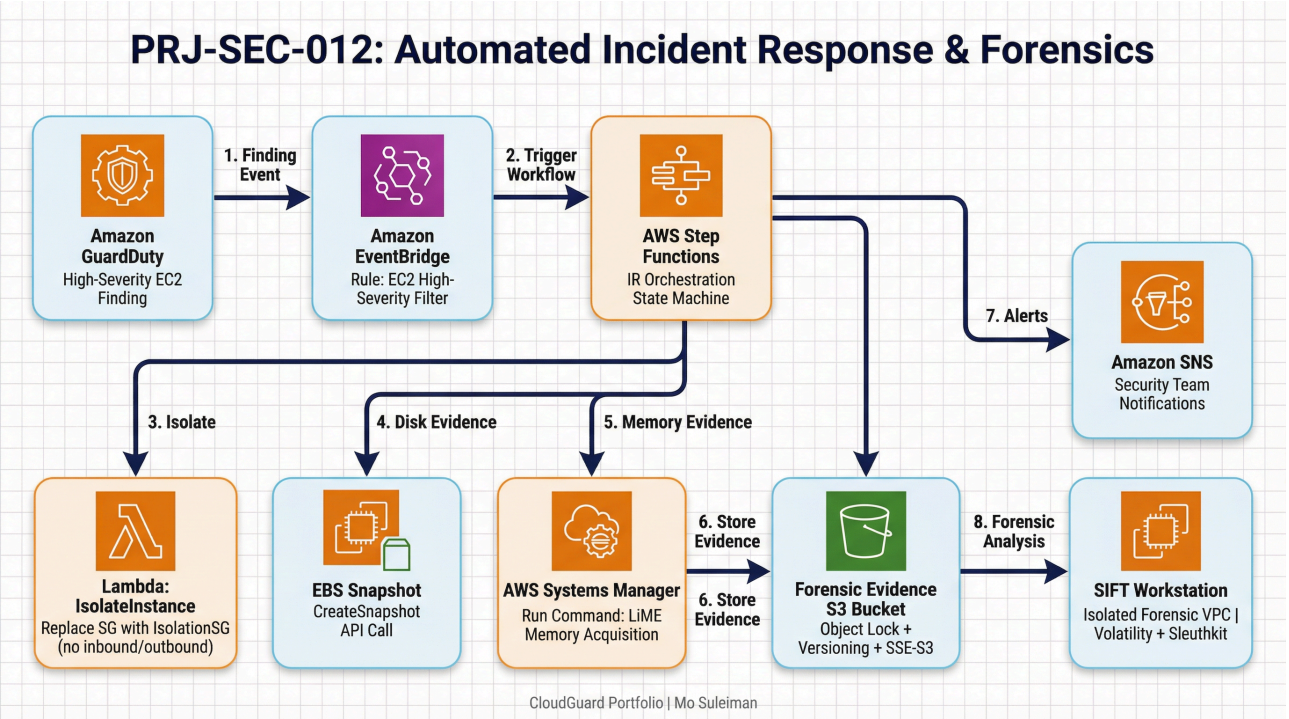
Framework	Control	Description
NIST CSF	RS.MI-1	Incidents are contained and mitigated.
NIST CSF	RS.AN-1	Forensic analysis is performed.
PCI-DSS	Req. 12.10.1	Create the incident response plan to be implemented in the event of system breach.
ISO 27001	A.16.1.7	Collection of evidence.
Business Value	—	Drastically reduces Mean Time to Contain (MTTC) from hours to seconds, prevents evidence spoliation, and provides a repeatable, auditable chain of custody for legal and compliance requirements.

Architecture Components

Component	Service	Purpose
Threat Simulation	AWS GuardDuty Tester	Generates real malicious traffic to trigger genuine GuardDuty findings.
Threat Detection	Amazon GuardDuty	Detects malicious activity on EC2 instances (e.g., cryptocurrency mining, C2 communication).
Event Routing	Amazon EventBridge	Filters high-severity GuardDuty findings and triggers the response workflow.
Orchestration	AWS Step Functions	State machine that coordinates the containment and evidence acquisition steps.
Containment	AWS Lambda	Modifies the compromised EC2 instance's security groups to block all inbound traffic and restrict outbound traffic to necessary endpoints.
Disk Forensics	Amazon EC2 / EBS	Creates an EBS snapshot of the compromised instance's root volume.
Memory Forensics	AWS Systems Manager	Uses Run Command to execute `avml` (Acquire Volatile Memory for Linux) and dump RAM to S3.
Evidence Storage	Amazon S3	Secure, immutable (Object Lock) storage for memory dumps and forensic artifacts.
Forensic Analysis	SANS SIFT Workstation	Isolated EC2 instance used by the analyst to investigate the disk and memory evidence.

---

2. Architecture Diagram



Workflow Steps:

1. **Threat Simulation:** The AWS GuardDuty Tester script runs on a target EC2 instance, generating real malicious DNS queries.
2. **Finding Event:** Amazon GuardDuty detects the malicious activity and generates a high-severity finding.
3. **Trigger Workflow:** An EventBridge rule, filtered for high-severity EC2 findings, triggers the Step Functions state machine.
4. **Isolate:** A Lambda function is invoked to replace the instance's security groups with an "IsolationSG", cutting off unauthorized network connectivity while allowing necessary forensic tools to communicate.
5. **Disk Evidence:** The state machine makes an API call to create a snapshot of the EBS volume attached to the instance.
6. **Memory Evidence:** The state machine uses SSM Run Command to execute a script that captures a full memory dump using `avml`.
7. **Store Evidence:** The memory image is uploaded directly to a highly restricted, immutable S3 bucket.

8. **Forensic Analysis:** An analyst attaches the snapshot and memory dump to an isolated SIFT Workstation to perform deep forensic analysis using Sleuthkit and Volatility.

---

3. Prerequisites

Before starting, confirm the following:

- You are logged into the AWS Console with an IAM user or role that has `AdministratorAccess`.
- You have an existing VPC with a public subnet and an Internet Gateway.
- You have a local terminal (or AWS Cloud9) with the AWS CLI, Git, Node.js, npm, AWS CDK, and Docker installed (required for the GuardDuty Tester).
- If using an Apple Silicon Mac (M1/M2/M3), ensure Docker Desktop has "Use Rosetta for x86_64/amd64 emulation" enabled.

---

4. Step-by-Step Deployment Guide

Step 4.1: Deploy the GuardDuty Tester Infrastructure First

We will deploy the tester first because it creates its own VPC. We need this VPC to create our Isolation Security Group in the correct network.

On your local machine (or AWS Cloud9), run the following commands:

```
git clone https://github.com/awslabs/amazon-guardduty-tester
cd amazon-guardduty-tester
npm install
cdk bootstrap aws://<YOUR_ACCOUNT_ID>/<YOUR_REGION>
cdk deploy
```

Type `y` to confirm the deployment. This takes about 5 minutes.

> ***Troubleshooting Note:*** *If `cdk deploy` fails with a Docker exit status 1 on Mac, ensure Docker is running, Rosetta emulation is enabled, and authenticate to ECR Public: `aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws`. Alternatively, use `cdk deploy --no-asset-bundling`.*

Step 4.2: Create the Isolation Security Group

1. Go to **EC2 -> Instances** and find the new instance named **`Driver-GuardDutyTester`** (or `Debian-GuardDutyTester`).
2. Note the **VPC ID** shown in the instance details (this is the tester's VPC).
3. Navigate to **EC2 -> Security Groups -> Create security group**.
4. **Name:** `IsolationSG-Tester`
5. **Description:** `Blocks inbound traffic, allows limited outbound for forensics.`
6. **VPC:** Select the **Tester VPC** (from step 2).
7. **Inbound Rules:** Delete all rules (no inbound traffic allowed).
8. **Outbound Rules:** Add the following rules to allow SSM and S3 communication:
    - **Type:** HTTPS | **Protocol:** TCP | **Port:** 443 | **Destination:** `pl-63a5400a` (S3 prefix list, find the one for your region)
    - **Type:** HTTPS | **Protocol:** TCP | **Port:** 443 | **Destination:** `0.0.0.0/0` (Allows SSM agent to reach the SSM endpoint)
9. Click **Create security group**. Note the Security Group ID.

Step 4.3: Create the Forensic Evidence S3 Bucket

1. Navigate to **S3 -> Create bucket**.
2. **Bucket name:** `forensic-evidence-store-bucket` (or a globally unique name).
3. **Block Public Access settings:** Ensure **Block all public access** is checked.
4. **Bucket Versioning:** Enable.
5. **Default encryption:** Enable with SSE-S3.
6. Click **Create bucket**.

Step 4.4: Create IAM Roles and Policies

1. **Update the Tester Instance Role:**
    - Navigate to **IAM -> Roles** and find the role created by the CDK stack (e.g., `GuardDutyTesterStack-Role...`).
    - Click **Add permissions -> Create inline policy**.
    - Add the following policy to allow the instance to upload memory dumps to S3:

```
{
 "Version": "2012-10-17",
 "Statement": [
     {
         "Effect": "Allow",
         "Action": [
             "s3:PutObject",
             "s3:PutObjectAcl"
         ],
         "Resource": "arn:aws:s3:::forensic-evidence-store-
bucket/*"
     }
 ]
}
```

    - Name the policy `ForensicsS3Upload` and click **Create policy**.
2. **Create the Lambda Execution Role:**
    - Create a new role for **Lambda**.
    - Attach the `AWSLambdaBasicExecutionRole` managed policy.
    - Add an inline policy named `LambdaEC2ModifyPolicy`:

```
{
 "Version": "2012-10-17",
 "Statement": [
     {
         "Effect": "Allow",
         "Action": "ec2:ModifyInstanceAttribute",
         "Resource": "*"
     }
 ]
}
```

- **Role name:** `Lambda-Isolation-Role`. Click **Create role**.

Step 4.5: Create the Isolation Lambda Function

1. Navigate to **Lambda -> Create function**.
2. **Function name:** `IsolateEC2Instance`
3. **Runtime:** Python 3.12
4. **Execution role:** Use existing role -> `Lambda-Isolation-Role`.
5. Paste the following Python code:

```
import boto3
import os

ec2 = boto3.client('ec2')
ISOLATION_SG_ID = os.environ['ISOLATION_SG_ID']

def lambda_handler(event, context):
    try:
        # Extract Instance ID from the GuardDuty finding via EventBridge
        instance_id = event['detail']['resource']['instanceDetails']
['instanceId']

        # Replace existing SGs with the Isolation SG
        response = ec2.modify_instance_attribute(
            InstanceId=instance_id,
            Groups=[ISOLATION_SG_ID]
        )

        return {
            'statusCode': 200,
            'body': f"Successfully isolated instance {instance_id}"
        }
    except Exception as e:
        print(f"Error isolating instance: {str(e)}")
        raise e
```

1. Under **Configuration -> Environment variables**, add a variable named `ISOLATION_SG_ID` and paste the ID of your `IsolationSG-Tester`.
2. Click **Deploy**.

Step 4.6: Create the Step Functions State Machine

1. **Create the Step Functions Role:**
    - Navigate to **IAM -> Roles -> Create role**.
    - **Trusted entity type:** AWS service -> Step Functions.
    - Add an inline policy named `StepFunctionsIRPolicy`:

```
{
 "Version": "2012-10-17",
 "Statement": [
     {
         "Effect": "Allow",
         "Action": "lambda:InvokeFunction",
         "Resource": "<ARN_OF_YOUR_ISOLATION_LAMBDA>"
     },
     {
         "Effect": "Allow",
         "Action": "ec2:CreateSnapshot",
         "Resource": "*"
     },
     {
         "Effect": "Allow",
         "Action": "ssm:SendCommand",
         "Resource": [
             "arn:aws:ec2:*:*:instance/*",
             "arn:aws:ssm:*:*:document/AWS-RunShellScript"
         ]
     }
 ]
}
```

- **Role name:** `StepFunctions-IR-Role`. Click **Create role**.

2. Navigate to **Step Functions -> State machines -> Create state machine**.
3. Choose **Blank**.
4. Click on the **Code** tab and paste the following ASL (Amazon States Language) definition. *Replace the placeholders with your actual Lambda ARN and S3 bucket name.*

> ***Important Note on Tags:*** *The* `CreateEBSSnapshot` *step reads the* `RootVolumeId` *tag to know which volume to snapshot. In this ASL, we read* `tags[4]` *or* `tags[6]` *depending on the exact tag order in the GuardDuty finding event payload. Ensure the* `RootVolumeId` *tag is present on the EC2 instance and its index matches the ASL configuration.*

```
{
  "Comment": "Incident Response Workflow for EC2",
  "StartAt": "IsolateInstance",
  "States": {
    "IsolateInstance": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "<ARN_OF_YOUR_ISOLATION_LAMBDA>",
        "Payload.$": "$"
      },
      "ResultPath": "$.isolationResult",
      "Next": "CreateEBSSnapshot"
    },
    "CreateEBSSnapshot": {
      "Type": "Task",
      "Resource": "arn:aws:states:::aws-sdk:ec2:createSnapshot",
      "Parameters": {
        "VolumeId.$": "$.detail.resource.instanceDetails.tags[4].value",
        "Description": "Forensic Snapshot triggered by GuardDuty"
      },
      "ResultPath": "$.snapshotResult",
      "Next": "AcquireMemory"
    },
    "AcquireMemory": {
      "Type": "Task",
      "Resource": "arn:aws:states:::aws-sdk:ssm:sendCommand",
      "Parameters": {
        "DocumentName": "AWS-RunShellScript",
        "InstanceIds.$":
"States.Array($.detail.resource.instanceDetails.instanceId)",
        "Parameters": {
          "commands": [
            "#!/bin/bash",
            "set -e",
            "cd /tmp",
            "curl -sL
https://github.com/microsoft/avml/releases/download/v0.14.0/avml -o avml",
            "chmod +x avml",
            "HOSTNAME=$(hostname)",
            "TIMESTAMP=$(date +%s)",
            "sudo ./avml /tmp/memory.mem",
            "aws s3 cp /tmp/memory.mem s3://forensic-evidence-store-
bucket/memorydumps/$HOSTNAME-$TIMESTAMP.mem",
            "rm -f /tmp/memory.mem /tmp/avml"
```

```
            ]
          }
        },
        "End": true
      }
    }
}
```

1. Click **Next**. Name it `Automated-IR-Workflow`.
2. Under **Permissions**, select **Choose an existing role** and select `StepFunctions-IR-Role`.
3. Click **Create state machine**.

Step 4.7: Enable Amazon GuardDuty

1. Navigate to **GuardDuty -> Get Started**.
2. Click **Enable GuardDuty**.
3. GuardDuty will now begin monitoring your environment.

Step 4.8: Create the EventBridge Trigger

1. Navigate to **Amazon EventBridge -> Rules -> Create rule**.
2. **Name:** `GuardDuty-High-Severity-IR`
3. **Event bus:** default
4. **Rule type:** Rule with an event pattern. Click **Next**.
5. **Event pattern:** Scroll down to the JSON editor and paste:

```
{
 "source": ["aws.guardduty"],
 "detail-type": ["GuardDuty Finding"],
 "detail": {
   "severity": [7, 7.0, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9,
8, 8.0, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9],
   "resource": {
     "resourceType": ["Instance"]
   }
 }
}
```

6. Click **Next**.
7. **Target types:** AWS service -> Step Functions state machine.
8. **State machine:** Select `Automated-IR-Workflow`.
9. Leave Execution role as “Create a new role for this specific resource”.
10. Click **Next** -> **Next** -> **Create rule**.

---

5. Testing the Workflow with the GuardDuty Tester

5.1 Prepare the Tester Instance

1. Go to **EC2 -> Instances** and find the instance named **`Driver-GuardDutyTester`** or `Debian-GuardDutyTester`.
2. Go to the **Storage** tab and copy the **Volume ID** (e.g., `vol-0abcd1234...`).
3. Go to the **Tags** tab -> Manage tags. Add a tag with **Key:** `RootVolumeId` and **Value:** `<Your Volume ID>`.

5.2 Generate Real Findings

1. Connect to the tester instance using Systems Manager (replace `<YOUR_REGION>`):

```
aws ssm start-session \
  --region <YOUR_REGION> \
  --document-name AWS-StartInteractiveCommand \
  --parameters command="cd /home/ssm-user/py_tester && bash -l" \
  --target $(aws ec2 describe-instances \
    --region <YOUR_REGION> \
    --filters "Name=tag:Name,Values=Debian-GuardDutyTester" \
    --query "Reservations[].Instances[?State.Name=='running'].InstanceId" \
    --output text)
```

1. Inside the SSM session, run the tester script to simulate cryptocurrency mining:

```
python3 guardduty_tester.py --finding 'CryptoCurrency:EC2/BitcoinTool.B!DNS'
```

5.3 Validate the Automated Response

1. Wait 5-15 minutes for GuardDuty to process the logs and generate the finding.
2. Check **Step Functions**: You should see a successful execution.
3. Check **EC2 Instances**: The target instance should now only have the `IsolationSG-Tester` attached.
4. Check **EC2 Snapshots**: A new snapshot of the root volume should exist.
5. Check **S3**: Your `forensic-evidence-store-bucket` bucket should contain a `.mem` file in the `memorydumps` folder.

---

6. SIFT Workstation Forensic Analysis Guide

Once the automated workflow completes, the forensic analyst takes over to investigate the collected evidence using the SANS SIFT (SANS Investigative Forensic Toolkit) Workstation.

6.1 Setting up the SIFT Workstation

1. Launch an EC2 instance using **Ubuntu Server 20.04 LTS** (t3.large, 50GB storage).

2. Place the instance in an **isolated Forensic VPC** with no internet access (or strict egress-only access for tool updates) to prevent cross-contamination.
3. Attach an IAM role to the SIFT instance that grants `s3:GetObject` access to your `forensic-evidence-store-bucket`.
4. SSH into the instance and install SIFT:

```
wget https://github.com/teamdfir/sift-cli/releases/latest/download/sift-cli-linux
chmod +x sift-cli-linux
sudo mv sift-cli-linux /usr/local/bin/sift
sudo sift install
```

6.2 Disk Forensics (Sleuthkit)

1. In the EC2 console, locate the EBS snapshot created by the Step Functions workflow.
2. Create a new EBS volume from this snapshot.
3. Attach the new volume to your SIFT Workstation as `/dev/sdf`.
4. SSH into the SIFT Workstation and mount the volume **read-only** to preserve evidence integrity:

```
sudo mkdir -p /mnt/evidence
sudo mount -o ro,noload /dev/xvdf1 /mnt/evidence
```

5. Use **Sleuthkit** to analyze the filesystem. For example, to list deleted files and directories:

```
fls -r -d /dev/xvdf1
```

6. To extract a specific deleted file using its inode number (e.g., inode 12345):

```
icat /dev/xvdf1 12345 > /tmp/recovered_file.txt
```

6.3 Memory Forensics (Volatility 3)

1. Download the memory dump from your S3 bucket to the SIFT Workstation:

```
aws s3 cp s3://forensic-evidence-store-
bucket/memorydumps/<filename>.mem /tmp/memory.mem
```

2. Use **Volatility 3** (pre-installed on SIFT) to analyze the RAM dump. First, list the running processes at the time of the incident:

```
vol -f /tmp/memory.mem linux.pslist
```

3. Check for active network connections to identify potential Command and Control (C2) communication:

```
vol -f /tmp/memory.mem linux.netstat
```

4. Look for hidden or injected malicious code in memory:

```
vol -f /tmp/memory.mem linux.malfind
```

5. Extract command-line arguments used by processes to understand attacker behavior:

```
vol -f /tmp/memory.mem linux.bash
```

---

7. Security Best Practices

- **Chain of Custody:** The entire automated process is logged by Step Functions and CloudTrail, providing an auditable chain of custody.
- **Evidence Integrity:** Storing evidence in an S3 bucket with Object Lock ensures it cannot be tampered with. Mounting the EBS volume as read-only prevents the analyst from modifying the evidence.
- **Outbound Connectivity:** The Isolation Security Group correctly restricts outbound traffic to necessary endpoints (S3 and SSM) via HTTPS (port 443), preventing the compromised instance from communicating with external C2 servers while allowing forensic tools to operate.
- **Pre-compiled Tools:** Using `avml` (a statically compiled binary) is superior to compiling LiME on the target instance. It avoids dependencies on kernel headers, which may not be available on cloud instances, and ensures a faster, more reliable memory acquisition process.

---

8. Cleanup

To avoid ongoing charges, delete the resources in this order:

1. **GuardDuty Tester:** From your local terminal, run `cdk destroy` inside the `amazon-guardduty-tester` directory.
2. **EventBridge:** Delete the `GuardDuty-High-Severity-IR` rule.
3. **Step Functions:** Delete the `Automated-IR-Workflow` state machine.
4. **Lambda:** Delete the `IsolateEC2Instance` function.
5. **EC2:** Terminate the SIFT Workstation. Delete the `IsolationSG-Tester` security group.
6. **EBS:** Delete the EBS snapshot and any forensic volumes created during testing.
7. **S3:** Empty and delete the `forensic-evidence-store-bucket` bucket.
8. **GuardDuty:** Suspend or disable GuardDuty if no longer needed.

9. **IAM:** Delete the roles created for Lambda, Step Functions, and the EC2 instance profile.

References

[1] https://github.com/awslabs/amazon-guardduty-tester “Amazon GuardDuty Tester” [2] https://github.com/teamdfir/sift-cli “SANS Investigative Forensic Toolkit (SIFT) Workstation” [3] https://github.com/microsoft/avml “Acquire Volatile Memory for Linux (avml)”