25541 - Cloud Computing & Security

King Abdullah II School of Engineering

Department of Computer Engineering

Cloud-Based Honeypot-as-a-Service

Authors:
Qabas Ahmad (ID: 20210786)
Malak Alshawish (ID: 20210944)

Supervisor:
Eng. Farah Abudabaseh

Date: May 24, 2025

Abstract

The rise of malicious cyber-activity is increasing exponentially across the Internet. Although firewalls and intrusion prevention systems offer a line of defense, honeypots serve as deceptive traps to gain insight into attacker behavior. This project proposes the development of a cloud-based honeypot system designed to attract, log, and analyze malicious attacker behavior in real time. The system uses Cowrie deployed on Amazon EC2. Attack data is collected and transferred to Amazon S3 and Amazon CloudWatch Logs, processed with AWS Lambda functions. The metadata of the attacker will be stored in Amazon DynamoDB and visualized using Amazon QuickSight, highlighting key cloud security principles, including network isolation, encryption, and IAM policies.

Contents

List of Figures

1 Introduction

1.1 Objectives

- Track and analyze attack behavior using AWS analytics tool.
- Demonstrate proactive security measures (IAM, encryption, secure architecture).
- Visualize attack patterns using collected metadata of the attacker.

1.2 Honeypot Definition

A honeypot is a computer security mechanism set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of information systems. Generally, a honeypot consists of data (for example, in a network site) that appears to be a legitimate part of the site which contains information or resources of value to attackers. It is actually isolated, monitored, and capable of blocking or analyzing the attackers. [4]

Honeypots can be differentiated based on whether they are physical or virtual:

- **Physical Honeypot**: a real machine with its own IP address, this machine simulates the behavior modeled by the system.

  Downside: High price of acquiring new machines, their maintenance and configuring specialized hardware.

- **Virtual Honeypot**: the use of this type of honeypot allows one to install and simulate hosts on the network from different operating systems.

  Downside: Simulate TCP/IP of the target operating system. However, this modality is still more frequent.

This project uses Cowrie Honeypot which is a medium interaction SSH and Telnet honeypot designed to log brute force attacks and shell interaction performed by an attacker. Cowrie also functions as an SSH and telnet proxy to observe attacker behavior to another system. [2] It can log brute-force attacks and an attacker's shell interaction. Cowrie is an open-source project developed by Michel Oosterhof. [6]

1.3 Cloud Integration

1.3.1 Cloud Computing

Cloud computing is on-demand access to computing resources— servers, data storage, networking capabilities, application development tools, software, AI-powered analytic platforms — over the internet with pay-per-use pricing. [5]

It provides a cost-effective, increased speed, unlimited scalability and enhanced stratgeic value compared to traditional on-premise solutions.

1.3.2 AWS Cloud

A cloud services provider (CSP) manages cloud-based technology services hosted at a remote data center and typically makes these resources available for a pay-as-you-go or monthly subscription fee.

Figure 1: Amazon Web Services

Amazon Web Services, Inc. (AWS) is a subsidiary of Amazon that provides on-demand cloud computing platforms and APIs to individuals on a metered, pay-as-you-go basis. Clients will often use this in combination with autoscaling.[1]

Autoscaling is a process that allows a client to use more computing in times of high application usage, and then scale down to reduce costs when there is less traffic.

The AWS Free Tier offers limited free access to many services for 12 months, including compute, storage, and databases, with usage limits. Some services also provide ”Always Free” tiers beyond the initial year, but exceeding usage thresholds incurs charges.

2 Cloud Architecture

2.1 AWS Services Used

Amazon Identity and Access Management (IAM): Manages secure access to AWS resources through granular permissions and roles, enabling least-privilege security policies.

Amazon Simple Storage Service (S3): Scalable object storage for data lakes, backups, and static websites, with 99.999999999% durability and versioning.

Amazon Elastic Compute Cloud (EC2): Virtual servers (instances) offering resizable compute capacity in the cloud, customizable by OS, storage, and networking.

Amazon DynamoDB: Fully managed NoSQL database for low-latency, high-throughput applications, with automatic scaling and serverless options.

Amazon CloudWatch: A service that monitors applications, responds to performance changes, optimizes resource use, and provides insights into operational health.

Amazon Virtual Private Cloud (VPC): a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud.

Amazon Lambda: Serverless compute service to run code without provisioning servers, scaling automatically and charging per millisecond of execution.

2.2 Implementation Diagram

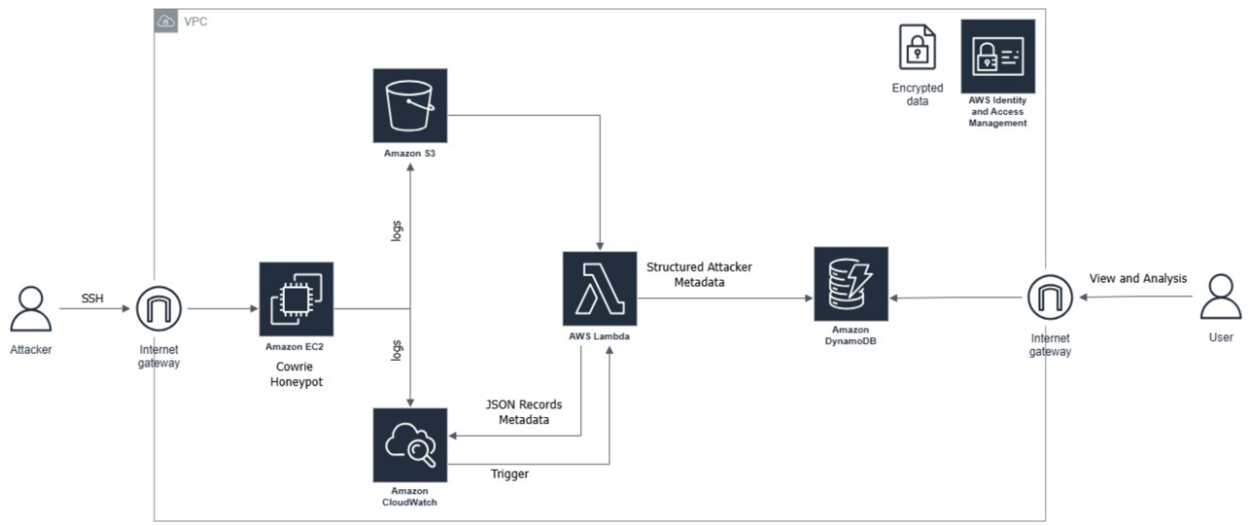


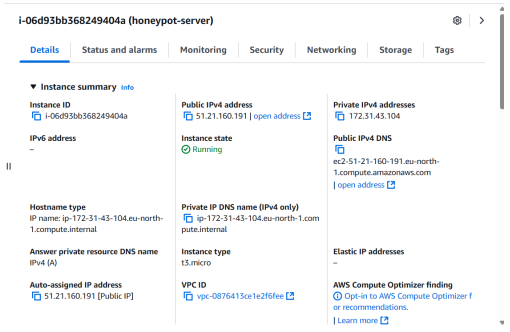
Figure 2: Honeypot-as-a-Service Implementation Diagram

The attacker will interact with the honeypot via SSH deployed on EC2 instances[3]. Both S3 storage and CloudWatch will be used to store and monitor log files which are analyzed to extract useful information using an automated Lambda function.

3 Core Implementation

3.1 Deploy EC2 Instance

An Amazon EC2 instance named honeypot-server was set up using Ubuntu 22. 04 and a t3. micro type for cost efficiency. A key pair enabled secure SSH authentication. A public IP address was assigned for easy access, and a security group was created to restrict SSH access to the administrator's IP. Cowrie was configured on port 2222 for attacker access.



i-06d93bb368249404a (honeypot-server)

Instance summary

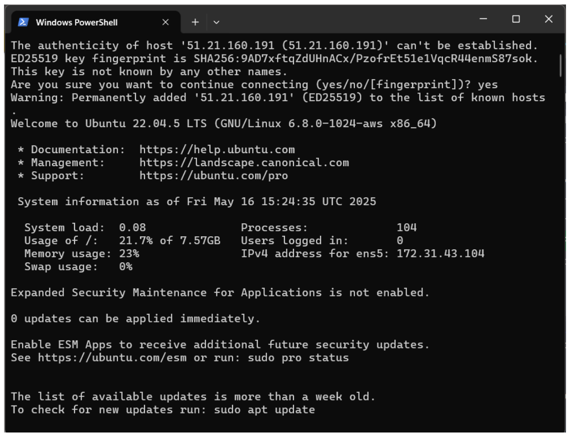
Instance ID i-06d93bb368249404a	Public IPv4 address 51.21.160.191 | open address	Private IPv4 addresses 172.31.43.104
IPv6 address –	Instance state Running	Public IPv4 DNS ec2-51-21-160-191.eu-north-1.compute.amazonaws.com | open address
Hostname type IP name: ip-172-31-43-104.eu-north-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-43-104.eu-north-1.compute.internal	
Answer private resource DNS name IPv4 (A)	Instance type t3.micro	Elastic IP addresses –
Auto-assigned IP address 51.21.160.191 [Public IP]	VPC ID vpc-0876413ce1e2f6fee	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Figure 3: EC2 Instance Configuration

3.2 SSH and Honeypot Setup

In order to access the EC2 instance and configure the honeypot, an SSH was performed to remotely connect to the instance using the the access key permissions and public IP address.

```
$ ssh -i "C:\Users\walki\OneDrive\Desktop\HoneyPot-KP.pem" ubuntu@51.21.160.191
```

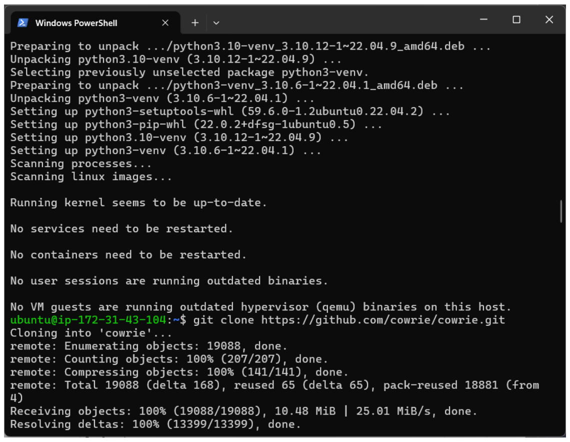


```
The authenticity of host '51.21.160.191 (51.21.160.191)' can't be established.
ED25519 key fingerprint is SHA256:9AD7xftqZdUHnACx/PzofrEt51e1VqcR44enmS87sok.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '51.21.160.191' (ED25519) to the list of known hosts
.
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1024-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri May 16 15:24:35 UTC 2025

  System load:  0.08               Processes:             104
  Usage of /:   21.7% of 7.57GB    Users logged in:       0
  Memory usage: 23%                IPv4 address for ens5: 172.31.43.104
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
```

Figure 4: SSH from local machine to the EC2 Instance

Then the packages lists for updates/installations are updated. Git and Python virtual environment tools are installed, and Cowrie honeypot code from GitHub is downloaded.

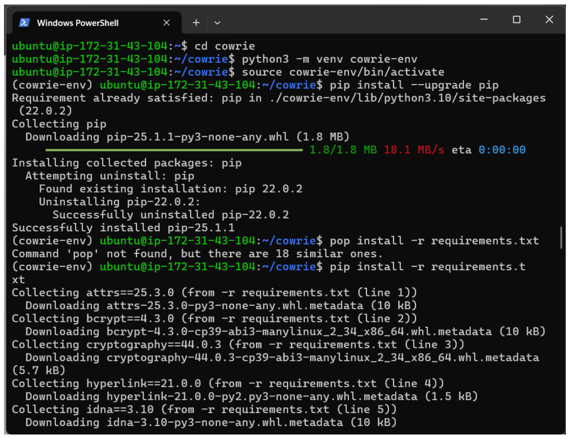
```
$ sudo apt install git python3-venv -y
$ git clone https://github.com/cowrie/cowrie.git
```



```
Preparing to unpack .../python3.10-venv_3.10.12-1~22.04.9_amd64.deb ...
Unpacking python3.10-venv (3.10.12-1~22.04.9) ...
Selecting previously unselected package python3-venv.
Preparing to unpack .../python3-venv_3.10.6-1~22.04.1_amd64.deb ...
Unpacking python3-venv (3.10.6-1~22.04.1) ...
Setting up python3-setuptools-whl (59.6.0-1.2ubuntu0.22.04.2) ...
Setting up python3-pip-whl (22.0.2+dfsg-1ubuntu0.5) ...
Setting up python3.10-venv (3.10.12-1~22.04.9) ...
Setting up python3-venv (3.10.6-1~22.04.1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-43-104:~$ git clone https://github.com/cowrie/cowrie.git
Cloning into 'cowrie'...
remote: Enumerating objects: 19088, done.
remote: Counting objects: 100% (207/207), done.
remote: Compressing objects: 100% (141/141), done.
remote: Total 19088 (delta 168), reused 65 (delta 65), pack-reused 18881 (from
4)
Receiving objects: 100% (19088/19088), 10.48 MiB | 25.01 MiB/s, done.
Resolving deltas: 100% (13399/13399), done.
```

Figure 5: Installing dependencies and downloading Cowrie from GitHub

```
$ cd cowrie
$ python3 -m venv cowrie-env
$ source cowrie-env/bin/activate
```

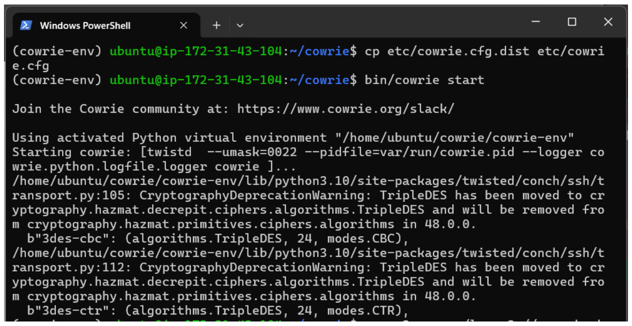


```
ubuntu@ip-172-31-43-104:~$ cd cowrie
ubuntu@ip-172-31-43-104:~/cowrie$ python3 -m venv cowrie-env
ubuntu@ip-172-31-43-104:~/cowrie$ source cowrie-env/bin/activate
(cowrie-env) ubuntu@ip-172-31-43-104:~/cowrie$ pip install --upgrade pip
Requirement already satisfied: pip in ./cowrie-env/lib/python3.10/site-packages
 (22.0.2)
Collecting pip
  Downloading pip-25.1.1-py3-none-any.whl (1.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 18.1 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.0.2
    Uninstalling pip-22.0.2:
      Successfully uninstalled pip-22.0.2
Successfully installed pip-25.1.1
(cowrie-env) ubuntu@ip-172-31-43-104:~/cowrie$ pop install -r requirements.txt
Command 'pop' not found, but there are 18 similar ones.
(cowrie-env) ubuntu@ip-172-31-43-104:~/cowrie$ pip install -r requirements.t
xt
Collecting attrs==25.3.0 (from -r requirements.txt (line 1))
  Downloading attrs-25.3.0-py3-none-any.whl.metadata (10 kB)
Collecting bcrypt==4.3.0 (from -r requirements.txt (line 2))
  Downloading bcrypt-4.3.0-cp39-abi3-manylinux_2_34_x86_64.whl.metadata (10 kB)
Collecting cryptography==44.0.3 (from -r requirements.txt (line 3))
  Downloading cryptography-44.0.3-cp39-abi3-manylinux_2_34_x86_64.whl.metadata
(5.7 kB)
Collecting hyperlink==21.0.0 (from -r requirements.txt (line 4))
  Downloading hyperlink-21.0.0-py2.py3-none-any.whl.metadata (1.5 kB)
Collecting idna==3.10 (from -r requirements.txt (line 5))
  Downloading idna-3.10-py3-none-any.whl.metadata (10 kB)
```

Figure 6: Creating separate, isolated Python environment for Cowrie

The default configuration template is prepared for customization, enabling tailored settings for the honeypot environment. Once configured, the service is launched, simulating vulnerable SSH/Telnet services to log and analyze incoming malicious activity.

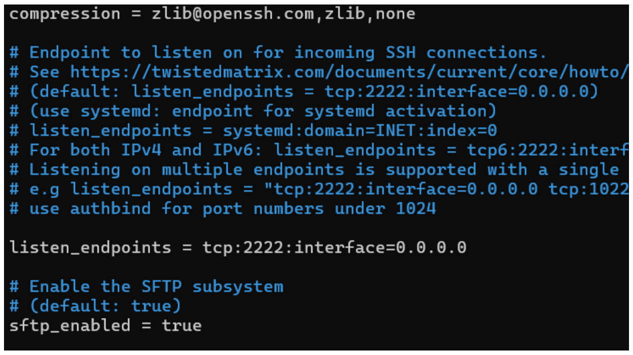
```
$ cp etc/cowrie.cfg.dist etc/cowrie.cfg
$ bin/cowrie start
```



```
(cowrie-env) ubuntu@ip-172-31-43-104:~/cowrie$ cp etc/cowrie.cfg.dist etc/cowri
e.cfg
(cowrie-env) ubuntu@ip-172-31-43-104:~/cowrie$ bin/cowrie start

Join the Cowrie community at: https://www.cowrie.org/slack/

Using activated Python virtual environment "/home/ubuntu/cowrie/cowrie-env"
Starting cowrie: [twistd  --umask=0022 --pidfile=var/run/cowrie.pid --logger co
wrie.python.logfile.logger cowrie ]...
/home/ubuntu/cowrie/cowrie-env/lib/python3.10/site-packages/twisted/conch/ssh/t
ransport.py:105: CryptographyDeprecationWarning: TripleDES has been moved to cr
yptography.hazmat.decrepit.ciphers.algorithms.TripleDES and will be removed fro
m cryptography.hazmat.primitives.ciphers.algorithms in 48.0.0.
  b"3des-cbc": (algorithms.TripleDES, 24, modes.CBC),
/home/ubuntu/cowrie/cowrie-env/lib/python3.10/site-packages/twisted/conch/ssh/t
ransport.py:112: CryptographyDeprecationWarning: TripleDES has been moved to cr
yptography.hazmat.decrepit.ciphers.algorithms.TripleDES and will be removed fro
m cryptography.hazmat.primitives.ciphers.algorithms in 48.0.0.
  b"3des-ctr": (algorithms.TripleDES, 24, modes.CTR),
```

Figure 7: Staring Cowrie on Device

Upon copying the configuration, a nano command was executed to change the SSH configurations of cowrie to accept vulnerable attacks at port 2222.



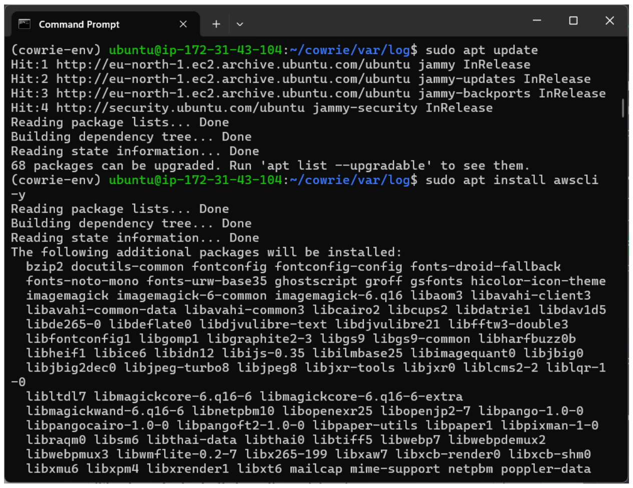
```
compression = zlib@openssh.com,zlib,none

# Endpoint to listen on for incoming SSH connections.
# See https://twistedmatrix.com/documents/current/core/howto/
# (default: listen_endpoints = tcp:2222:interface=0.0.0.0)
# (use systemd: endpoint for systemd activation)
# listen_endpoints = systemd:domain=INET:index=0
# For both IPv4 and IPv6: listen_endpoints = tcp6:2222:interf
# Listening on multiple endpoints is supported with a single
# e.g listen_endpoints = "tcp:2222:interface=0.0.0.0 tcp:1022
# use authbind for port numbers under 1024

listen_endpoints = tcp:2222:interface=0.0.0.0

# Enable the SFTP subsystem
# (default: true)
sftp_enabled = true
```

Figure 8: Cowrie Configuration File

3.3 Send Logs to S3

To Set up AWS Command Line Interface (CLI), the package manager updates its repository metadata. The AWS (CLI) is then installed to enable programmatic interaction with AWS services, such as configuring cloud resources or managing the honeypot instance directly from the terminal.

```
$ sudo apt update
$ sudo apt install awscli -y
```



```
(cowrie-env) ubuntu@ip-172-31-43-104:~/cowrie/var/log$ sudo apt update
Hit:1 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
68 packages can be upgraded. Run 'apt list --upgradable' to see them.
(cowrie-env) ubuntu@ip-172-31-43-104:~/cowrie/var/log$ sudo apt install awscli
-y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bzip2 docutils-common fontconfig fontconfig-config fonts-droid-fallback
  fonts-noto-mono fonts-urw-base35 ghostscript groff gsfonts hicolor-icon-theme
  imagemagick imagemagick-6-common imagemagick-6.q16 libaom3 libavahi-client3
  libavahi-common-data libavahi-common3 libcairo2 libcups2 libdatrie1 libdav1d5
  libde265-0 libdeflate0 libdjvulibre-text libdjvulibre21 libfftw3-double3
  libfontconfig1 libgomp1 libgraphite2-3 libgs9 libgs9-common libharfbuzz0b
  libheif1 libice6 libidn12 libijs-0.35 libilmbase25 libimagequant0 libjbig0
  libjbig2dec0 libjpeg-turbo8 libjpeg8 libjxr-tools libjxr0 liblcms2-2 liblqr-1
-0
  libltdl7 libmagickcore-6.q16-6 libmagickcore-6.q16-6-extra
  libmagickwand-6.q16-6 libnetpbm10 libopenexr25 libopenjp2-7 libpango-1.0-0
  libpangocairo-1.0-0 libpangoft2-1.0-0 libpaper-utils libpaper1 libpixman-1-0
  libraqm0 libsm6 libthai-data libthai0 libtiff5 libwebp7 libwebpdemux2
  libwebpmux3 libwmflite-0.2-7 libx265-199 libxaw7 libxcb-render0 libxcb-shm0
  libxmu6 libxpm4 libxrender1 libxt6 mailcap mime-support netpbm poppler-data
```

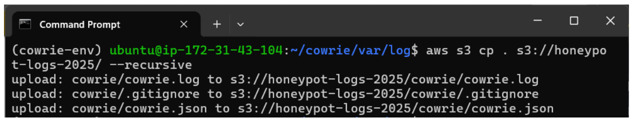
Figure 9: Updating repository metadata and installing the AWS CLI

To configure the AWS CLI, run aws configure and provide your Access Key ID and Secret Access Key when prompted, specify the region as eu-north-1.

```
$ aws configure
```

All files and subdirectories from the current local directory are uploaded to the AWS S3 bucket honeypot-logs-2025, backing up the honeypot logs to cloud storage.

```
$ aws s3 cp . s3://honeypot-logs-2025/ --recursive
```



```
(cowrie-env) ubuntu@ip-172-31-43-104:~/cowrie/var/log$ aws s3 cp . s3://honeypo
t-logs-2025/ --recursive
upload: cowrie/cowrie.log to s3://honeypot-logs-2025/cowrie/cowrie.log
upload: cowrie/.gitignore to s3://honeypot-logs-2025/cowrie/.gitignore
upload: cowrie/cowrie.json to s3://honeypot-logs-2025/cowrie/cowrie.json
```

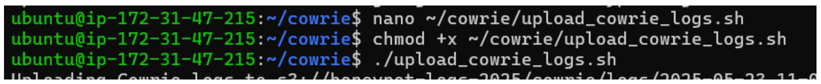
Figure 10: Logs Uploaded to AWS S3 Instance

A Bash script was written to automate the archival process of Cowrie honeypot logs and captured attacker downloads to a designated AWS S3 bucket (honeypot-logs-2025). Key operations include:

- Timestamp Generation: A unique timestamp is generated to organize the backups chronologically.
- Path Definitions: Local source directories for logs (˜/cowrie/var/log/cowrie/) and downloads (˜/cowrie/var/lib/cowrie/downloads/) are mapped to S3 destination paths prefixed with the timestamp.
- AWS CLI Execution: The aws s3 copy command recursively uploads the files to their respective S3 paths.

```
#!/bin/bash


bucket="honeypot-logs-2025"
timestamp=$ (date +"%Y-%m-%d_%H-%M-%S")

logs_src="$ home/cowrie/var/log/cowrie/"
downloads_src="$ home/cowrie/var/lib/cowrie/downloads/"

logs_dest="s3://$ bucket/cowrie/logs/$ timestamp/"
downloads_dest="s3://$ bucket/cowrie/downloads/$ timestamp/"


echo "Uploading Cowrie logs to $ logs_dest ..."
aws s3 cp "$ logs_src" "$ logs_dest" --recursive

echo "Uploading Cowrie downloads to $ downloads_dest ..."
aws s3 cp "$ downloads_src" "$ downloads_dest" --recursive

echo "Upload complete!"
```

After changing the permissions, running this script will automatically upload the scripts from the honeypot logs to the S3 instance.



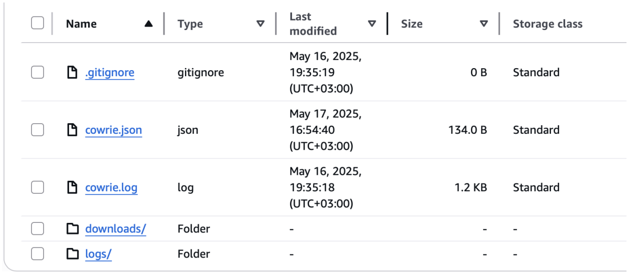
```
ubuntu@ip-172-31-47-215:~/cowrie$ nano ~/cowrie/upload_cowrie_logs.sh
ubuntu@ip-172-31-47-215:~/cowrie$ chmod +x ~/cowrie/upload_cowrie_logs.sh
ubuntu@ip-172-31-47-215:~/cowrie$ ./upload_cowrie_logs.sh
```

Figure 11: Creating and Running the Shell Script

3.4 S3 bucket - log storage

We configured the S3 bucket **honeypot-logs-2025** to include the following objects:

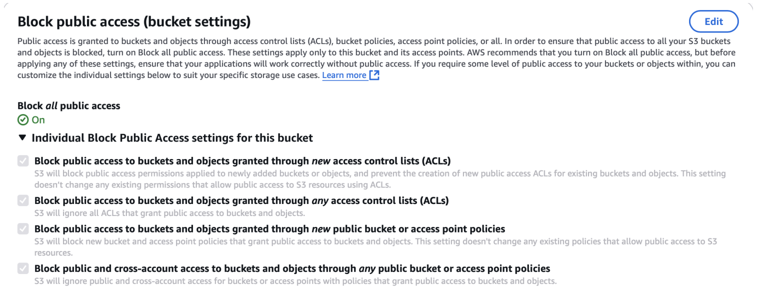
- **cowrie.log**: which is the main log file generated by the honeypot.
- **cowrie.json**: contains the structured JSON format for easier parsing and analysis.
- **.gitignore**: for Git-based sync.
- **downloads/**: to store files that the attacker tried to download/upload via honeypot.
- **logs/**: to store archived logs.



	Name ▲	Type ▽	Last modified ▽	Size ▽	Storage class
☐	.gitignore	gitignore	May 16, 2025, 19:35:19 (UTC+03:00)	0 B	Standard
☐	cowrie.json	json	May 17, 2025, 16:54:40 (UTC+03:00)	134.0 B	Standard
☐	cowrie.log	log	May 16, 2025, 19:35:18 (UTC+03:00)	1.2 KB	Standard
☐	downloads/	Folder	-	-	-
☐	logs/	Folder	-	-	-

Figure 12: S3 bucket objects

Permissions were set to block all public all public access to prevent our logs from being exposed to the public. This ensures that only our EC2 instance and our Lambda function can access the data.



Block public access (bucket settings)

Edit

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

Block *all* public access
✓ On

▼ **Individual Block Public Access settings for this bucket**

✓ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

✓ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.

✓ **Block public access to buckets and objects granted through *new* public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

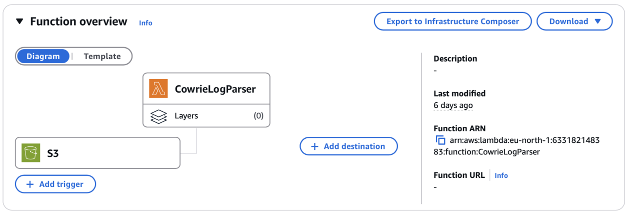
✓ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Figure 13: S3 blocked premissions

3.5 AWS Lambda - Log Processing

A serverless function, **CowrieLogParser**, was created to automate the parsing of the honeypot logs and storing them in a DynamoDB table. It is triggered by S3 **ObjectCreated** event which is invoked whenever a new file is uploaded to the S3 bucket and uses an IAM role with:

- **S3:** GetObject on honeypot-logs-2025
- **DynamoDB:** PutItem on CowrieLogs
- **Logs:** CloudWatch logging



Description
-

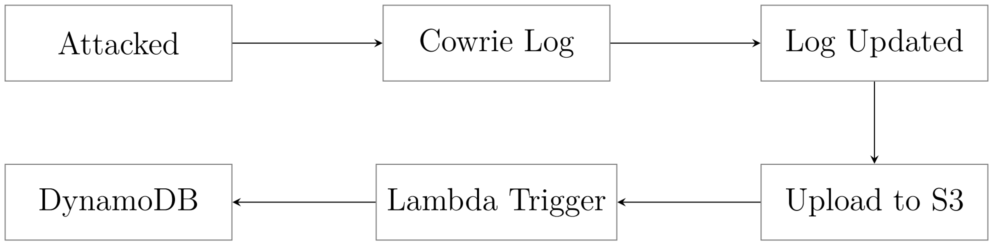
Last modified
6 days ago

Function ARN
arn:aws:lambda:eu-north-1:633182148383:function:CowrieLogParser

Function URL Info
-

Figure 14: S3 as a Lambda trigger

The typical flow of events thus far looks like this:



Parsed Log File Contents:

- **session:** unique session ID assigned by Cowrie
- **timestamp:** time of event occurrence
- **src-ip:** IP address of the attacker
- **input:** command entered by the attacker
- **eventid:** type of event(login, input etc)

The second function created, **LogDynamoDBToCloudWatch**, logs every item inserted in the DynamoDB table to CloudWatch logs. The purpose is to monitor attackers' activity in real-time, and set metric filters and alarms to stay in check. The function receives event data from the inserted item,, parses them and prints them to CloudWatch logs.

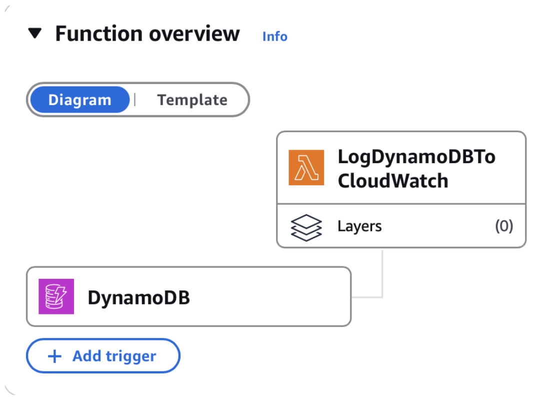
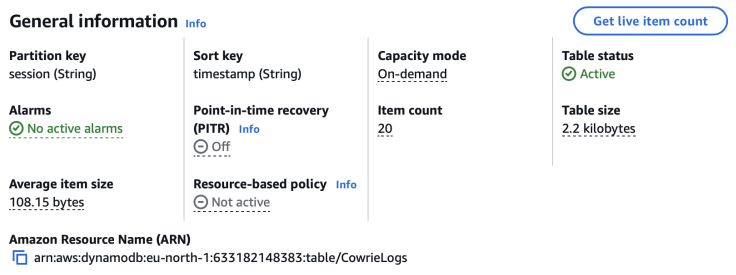


Figure 15: DynamoDB as a Lambda trigger

3.6 DynamoDB - Attacker log storage

The **CowrieLogs** table enables storage of parsed honeypot log data, with each item representing a single honeypot interaction. The Lambda function interacts directly in the table using **put-item()**.

The table is structured around a composite primary key consisting of a PartitionKey and a SortKey, which allows us to store multiple items under the same session and sorting by timestamp within each session.



Partition key
session (String)

Sort key
timestamp (String)

Capacity mode
On-demand

Table status
Active

Alarms
No active alarms

Point-in-time recovery (PITR) Info
Off

Item count
20

Table size
2.2 kilobytes

Average item size
108.15 bytes

Resource-based policy Info
Not active

Amazon Resource Name (ARN)
arn:aws:dynamodb:eu-north-1:633182148383:table/CowrieLogs

Figure 16: DynamoDB table information

Cowrie honeypot logs stored in DynamoDB, where each entry captures detailed metadata about SSH connection attempts.

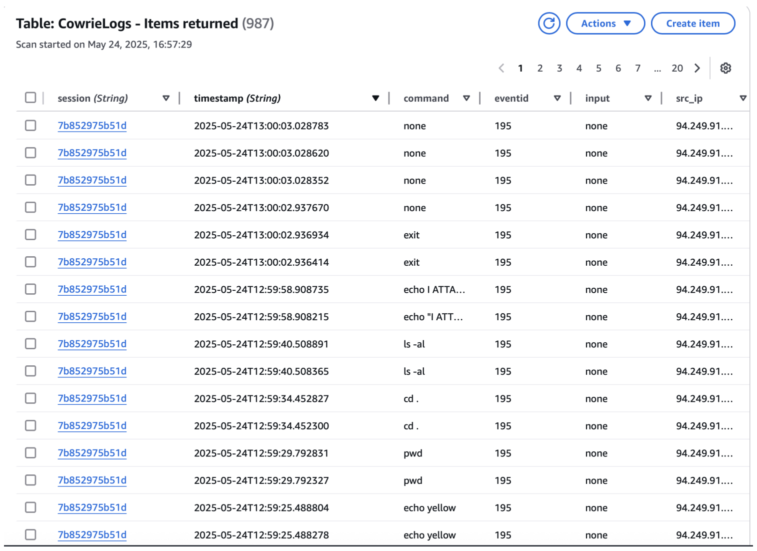
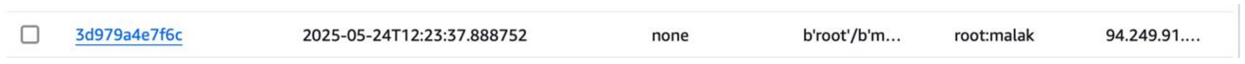


Table: CowrieLogs - Items returned (987)
Scan started on May 24, 2025, 16:57:29

session *(String)*	timestamp *(String)*	command	eventid	input	src_ip
7b852975b51d	2025-05-24T13:00:03.028783	none	195	none	94.249.91....
7b852975b51d	2025-05-24T13:00:03.028620	none	195	none	94.249.91....
7b852975b51d	2025-05-24T13:00:03.028352	none	195	none	94.249.91....
7b852975b51d	2025-05-24T13:00:02.937670	none	195	none	94.249.91....
7b852975b51d	2025-05-24T13:00:02.936934	exit	195	none	94.249.91....
7b852975b51d	2025-05-24T13:00:02.936414	exit	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:58.908735	echo I ATTA...	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:58.908215	echo "I ATT...	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:40.508891	ls -al	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:40.508365	ls -al	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:34.452827	cd .	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:34.452300	cd .	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:29.792831	pwd	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:29.792327	pwd	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:25.488804	echo yellow	195	none	94.249.91....
7b852975b51d	2025-05-24T12:59:25.488278	echo yellow	195	none	94.249.91....

Figure 17: Cowrie logs in DynamoDB

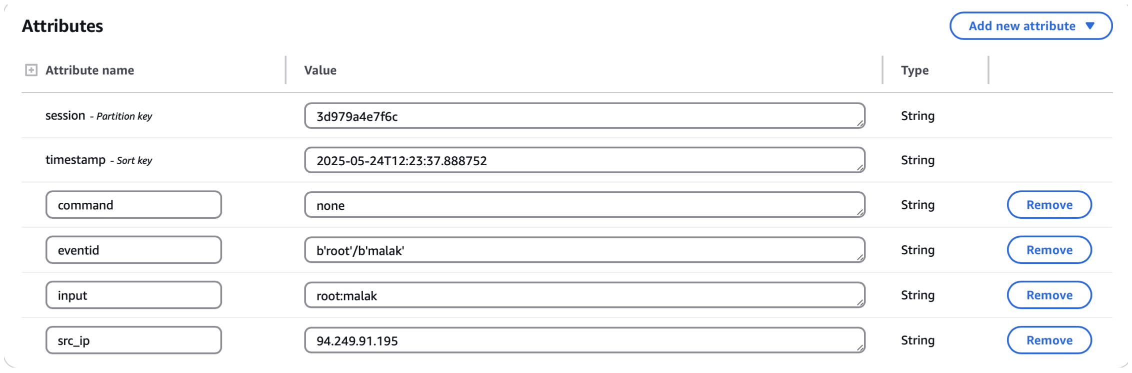
For example, this entry shows the attempt of the attacker to log in to the honyepot:



3d979a4e7f6c	2025-05-24T12:23:37.888752	none	b'root'/b'm...	root:malak	94.249.91....

Figure 18: Attempted login to the honeypot log

And it contains the following information in details.



Attributes

Attribute name	Value	Type
session - *Partition key*	3d979a4e7f6c	String
timestamp - *Sort key*	2025-05-24T12:23:37.888752	String
command	none	String
eventid	b'root'/b'malak'	String
input	root:malak	String
src_ip	94.249.91.195	String

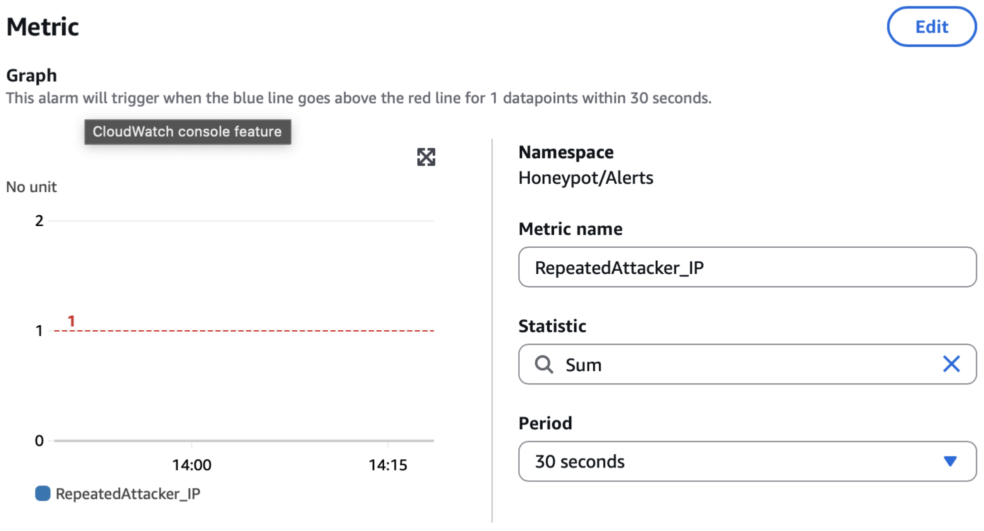
Figure 19: Log content

This shows the parsed data, confirming that our Lambda function is working properly.

3.7 Cloudwatch - Monitoring and Alerting

This is our central monitoring and alerting system in our setup. It connects DynamoDB and Lambda through logs and metrics to enable detection and response to attackers.

We set up an alarm called **AttackerSent**, with the IP address of the attacker as the alarm metric **RepeatedAttacker-IP**. This alarm is triggered if the specified IP address **94.249.91.195** appears at least once in the last 30 seconds.
It was also setup to send an SNS notification to the email address **malakshUni@gmail.com**.



Metric

Graph
This alarm will trigger when the blue line goes above the red line for 1 datapoints within 30 seconds.

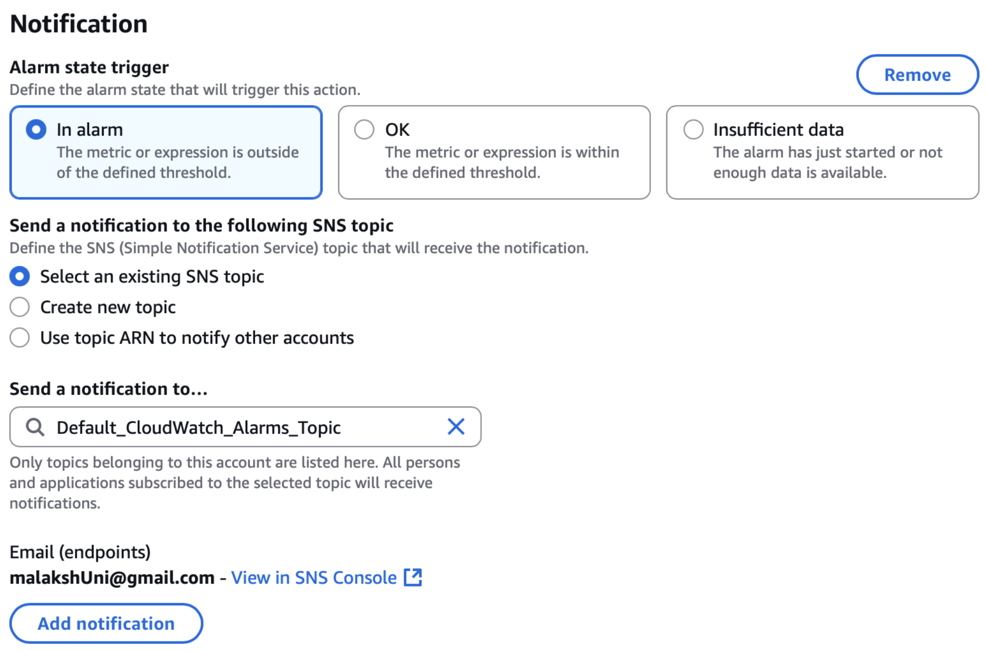
Namespace
Honeypot/Alerts

Metric name
RepeatedAttacker_IP

Statistic
Sum

Period
30 seconds

Figure 20: Metric setup



Notification

Alarm state trigger
Define the alarm state that will trigger this action.

[✓ In alarm] — The metric or expression is outside of the defined threshold.
OK — The metric or expression is within the defined threshold.
Insufficient data — The alarm has just started or not enough data is available.

Send a notification to the following SNS topic
Define the SNS (Simple Notification Service) topic that will receive the notification.

[✓ Select an existing SNS topic]
Create new topic
Use topic ARN to notify other accounts

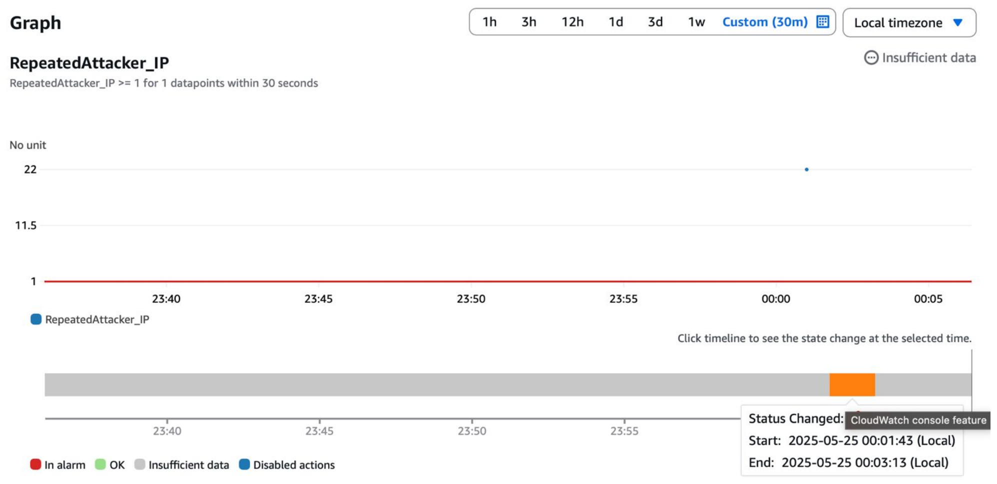
Send a notification to…
Default_CloudWatch_Alarms_Topic

Only topics belonging to this account are listed here. All persons and applications subscribed to the selected topic will receive notifications.

Email (endpoints)
malakshUni@gmail.com - View in SNS Console

Figure 21: Notification setup

An attempted analysis using a CloudWatch graph to visualize SSH login activity over time, aiming to detect unusual spikes that might indicate brute-force attempts. The graph provides a starting point for identifying trends and configuring SNS alerts to notify administrators when suspicious patterns emerge.

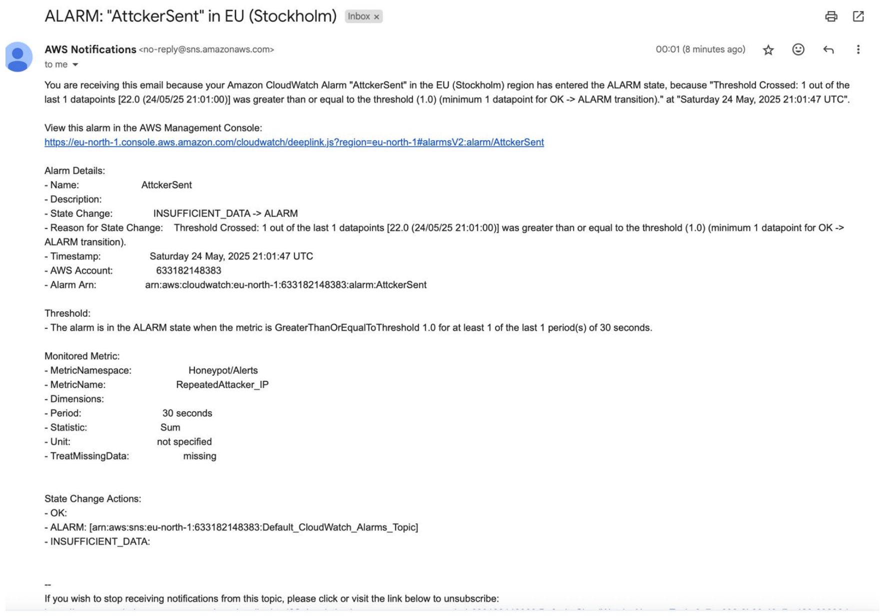


RepeatedAttacker_IP
RepeatedAttacker_IP >= 1 for 1 datapoints within 30 seconds

Insufficient data

Status Changed: CloudWatch console feature
Start: 2025-05-25 00:01:43 (Local)
End: 2025-05-25 00:03:13 (Local)

Figure 22: Attacker graph



ALARM: "AttckerSent" in EU (Stockholm) Inbox ×

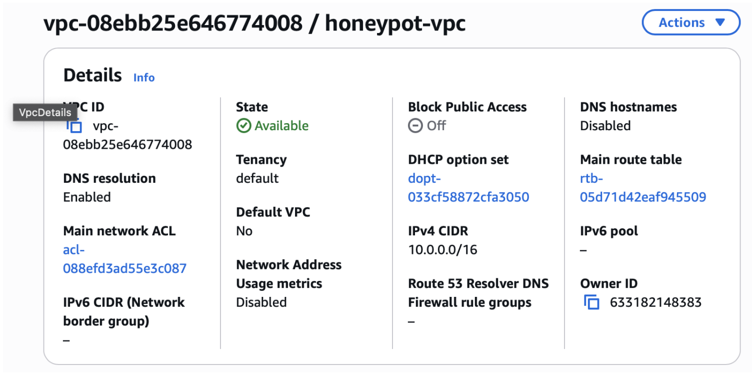
AWS Notifications <no-reply@sns.amazonaws.com>
to me
00:01 (8 minutes ago)

```
You are receiving this email because your Amazon CloudWatch Alarm "AttckerSent" in the EU (Stockholm) region has entered the ALARM state, because "Threshold Crossed: 1 out of the
last 1 datapoints [22.0 (24/05/25 21:01:00)] was greater than or equal to the threshold (1.0) (minimum 1 datapoint for OK -> ALARM transition)." at "Saturday 24 May, 2025 21:01:47 UTC".

View this alarm in the AWS Management Console:
https://eu-north-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=eu-north-1#alarmsV2:alarm/AttckerSent

Alarm Details:
- Name:                   AttckerSent
- Description:
- State Change:           INSUFFICIENT_DATA -> ALARM
- Reason for State Change:    Threshold Crossed: 1 out of the last 1 datapoints [22.0 (24/05/25 21:01:00)] was greater than or equal to the threshold (1.0) (minimum 1 datapoint for OK ->
ALARM transition).
- Timestamp:              Saturday 24 May, 2025 21:01:47 UTC
- AWS Account:            633182148383
- Alarm Arn:              arn:aws:cloudwatch:eu-north-1:633182148383:alarm:AttckerSent

Threshold:
- The alarm is in the ALARM state when the metric is GreaterThanOrEqualToThreshold 1.0 for at least 1 of the last 1 period(s) of 30 seconds.

Monitored Metric:
- MetricNamespace:              Honeypot/Alerts
- MetricName:                   RepeatedAttacker_IP
- Dimensions:
- Period:                       30 seconds
- Statistic:                    Sum
- Unit:                         not specified
- TreatMissingData:             missing


State Change Actions:
- OK:
- ALARM: [arn:aws:sns:eu-north-1:633182148383:Default_CloudWatch_Alarms_Topic]
- INSUFFICIENT_DATA:


--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
```

Figure 23: SNS notification

4 Security Analysis

Our instance is hosted in a restricted Virtual Private Network(VPC) to provide isolation and avoid any outbound traffic initiation. This ensures that the attacker can't use the honeypot to launch attacks outwards.

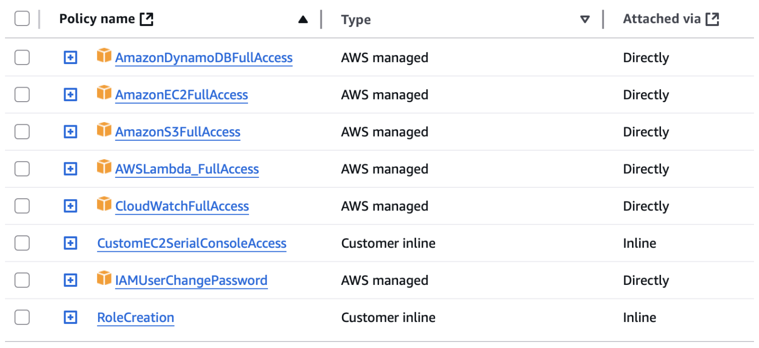


vpc-08ebb25e646774008 / honeypot-vpc

VPC ID: vpc-08ebb25e646774008	State: Available	Block Public Access: Off	DNS hostnames: Disabled
DNS resolution: Enabled	Tenancy: default	DHCP option set: dopt-033cf58872cfa3050	Main route table: rtb-05d71d42eaf945509
Main network ACL: acl-088efd3ad55e3c087	Default VPC: No	IPv4 CIDR: 10.0.0.0/16	IPv6 pool: –
IPv6 CIDR (Network border group): –	Network Address Usage metrics: Disabled	Route 53 Resolver DNS Firewall rule groups: –	Owner ID: 633182148383

Figure 24: VPC setup

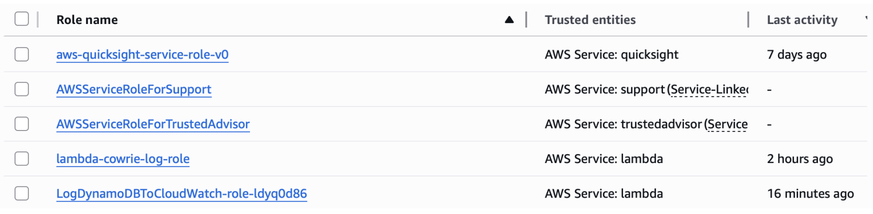
Moreover, the required IAM roles were setup based on the principles of least privilege, such that the permissions were just enough to finish the designated task. This was done through the root user account, and any extra permissions needed were granted incrementally. The **honeypot-admin** user had the following permissions granted:



Policy name	Type	Attached via
AmazonDynamoDBFullAccess	AWS managed	Directly
AmazonEC2FullAccess	AWS managed	Directly
AmazonS3FullAccess	AWS managed	Directly
AWSLambda_FullAccess	AWS managed	Directly
CloudWatchFullAccess	AWS managed	Directly
CustomEC2SerialConsoleAccess	Customer inline	Inline
IAMUserChangePassword	AWS managed	Directly
RoleCreation	Customer inline	Inline

Figure 25: honeypot-admin permissions

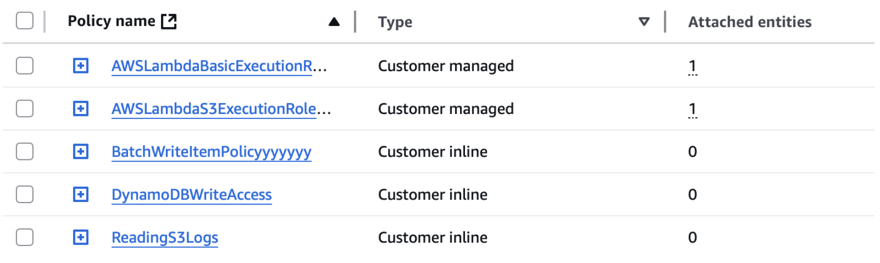
Two more roles were also configured over the three default, AWS-managed ones.



Role name	Trusted entities	Last activity
aws-quicksight-service-role-v0	AWS Service: quicksight	7 days ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linke	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service	-
lambda-cowrie-log-role	AWS Service: lambda	2 hours ago
LogDynamoDBToCloudWatch-role-ldyq0d86	AWS Service: lambda	16 minutes ago

Figure 26: Roles

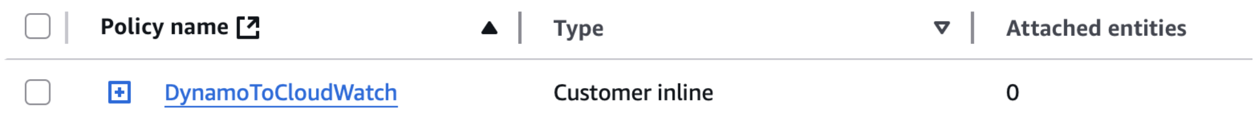
- **lambda-cowrie-log-role:** Reads from S3, writes to DynamoDB and logs to CloudWatch. The following permissions were attached:



Policy name	Type	Attached entities
AWSLambdaBasicExecutionR...	Customer managed	1
AWSLambdaS3ExecutionRole...	Customer managed	1
BatchWriteItemPolicyyyyyyy	Customer inline	0
DynamoDBWriteAccess	Customer inline	0
ReadingS3Logs	Customer inline	0

Figure 27: lambda-cowrie-log-role permissions

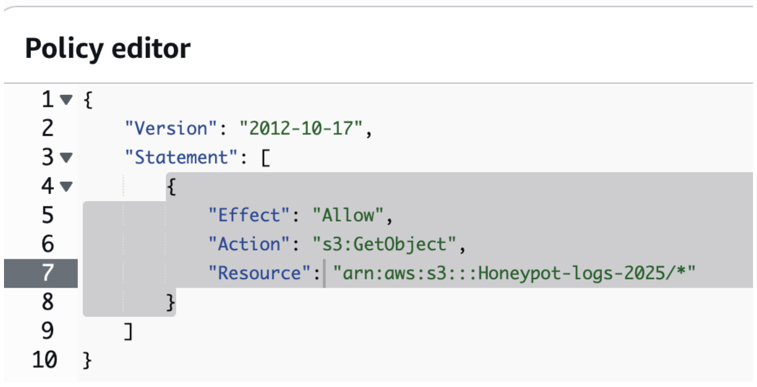
- **LogDynamoDBToCloudWatch-role-ldyq0d86:** Listens to DynamoDB streams, reads stream records, logs to CloudWatch. The following permissions were attached:



	Policy name	Type	Attached entities
	DynamoToCloudWatch	Customer inline	0

Figure 28: LogDynamoDBToCloudWatch-role-ldyq0d86 permissions

Each permission contains a JSON file that defines what actions the role is allowed or denied to perform, and on which resources. So for example, the **ReadingS3Logs** policy in *lambda-cowrie-log-role* looks like this:



Policy editor

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::Honeypot-logs-2025/*"
        }
    ]
}
```

Figure 29: Policy example

5 Evaluation and Testing

5.1 Functional and Security Testing

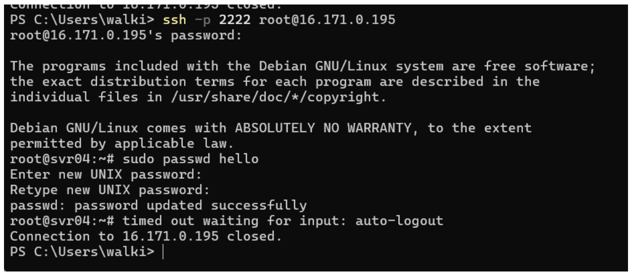
5.1.1 Direct SSH Access

To test the Cowrie honeypot, the service was first started using the terminal by navigating to the Cowrie directory, activating the virtual environment, and running the bin/cowrie start command.

Once active, Cowrie listens for SSH connections on port 2222. A simulated attack was performed locally by attempting to connect via SSH to Cowrie in the EC2 instance using:

```
$ ssh -p 2222 root@16.171.0.195
```

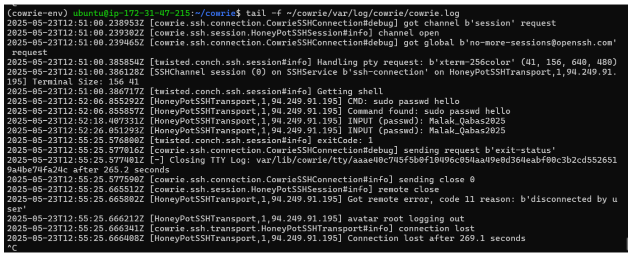
The command led to a fake shell environment designed to mimic a real system. Various common commands such as ls, pwd, and uname -a were entered to observe the honeypot's responses.



```
PS C:\Users\walki> ssh -p 2222 root@16.171.0.195
root@16.171.0.195's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@svr04:~# sudo passwd hello
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@svr04:~# timed out waiting for input: auto-logout
Connection to 16.171.0.195 closed.
PS C:\Users\walki> |
```

Figure 30: Cowrie's Fake Shell Environemnt with Commands

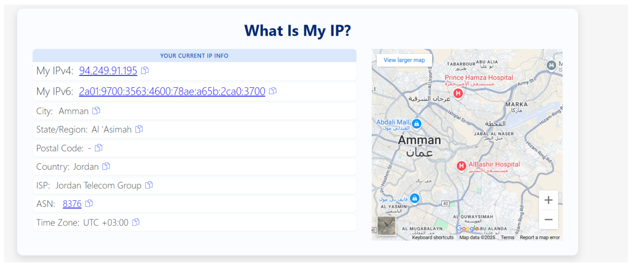
These actions, along with login attempts, were logged in detail by Cowrie and could be reviewed through cowrie.log and cowrie.json files. This allowed for easy analysis of attacker behavior. Once testing was complete, the honeypot was stopped using the bin/cowrie stop command. The testing confirmed that Cowrie effectively emulates a vulnerable system and logs malicious activity for further inspection.



```
(cowrie-env) ubuntu@ip-172-31-47-215:~/cowrie$ tail -f ~/cowrie/var/log/cowrie/cowrie.log
2025-05-23T12:51:00.238953Z [cowrie.ssh.connection.CowrieSSHConnection#debug] got channel b'session' request
2025-05-23T12:51:00.239302Z [cowrie.ssh.session.HoneyPotSSHSession#info] channel open
2025-05-23T12:51:00.239465Z [cowrie.ssh.connection.CowrieSSHConnection#debug] got global b'no-more-sessions@openssh.com'
 request
2025-05-23T12:51:00.385854Z [twisted.conch.ssh.session#info] Handling pty request: b'xterm-256color' (41, 156, 640, 480)
2025-05-23T12:51:00.386128Z [SSHChannel session (0) on SSHService b'ssh-connection' on HoneyPotSSHTransport,1,94.249.91.
195] Terminal Size: 156 41
2025-05-23T12:51:00.386717Z [twisted.conch.ssh.session#info] Getting shell
2025-05-23T12:52:06.855292Z [HoneyPotSSHTransport,1,94.249.91.195] CMD: sudo passwd hello
2025-05-23T12:52:06.855857Z [HoneyPotSSHTransport,1,94.249.91.195] Command found: sudo passwd hello
2025-05-23T12:52:18.407331Z [HoneyPotSSHTransport,1,94.249.91.195] INPUT (passwd): Malak_Qabas2025
2025-05-23T12:52:26.051293Z [HoneyPotSSHTransport,1,94.249.91.195] INPUT (passwd): Malak_Qabas2025
2025-05-23T12:55:25.576800Z [twisted.conch.ssh.session#info] exitCode: 1
2025-05-23T12:55:25.577016Z [cowrie.ssh.connection.CowrieSSHConnection#debug] sending request b'exit-status'
2025-05-23T12:55:25.577401Z [-] Closing TTY Log: var/lib/cowrie/tty/aaae40c745f5b0f10496c054aa49e0d364eabf00c3b2cd552651
9a4be74fa24c after 265.2 seconds
2025-05-23T12:55:25.577590Z [cowrie.ssh.connection.CowrieSSHConnection#info] sending close 0
2025-05-23T12:55:25.665512Z [cowrie.ssh.session.HoneyPotSSHSession#info] remote close
2025-05-23T12:55:25.665802Z [HoneyPotSSHTransport,1,94.249.91.195] Got remote error, code 11 reason: b'disconnected by u
ser'
2025-05-23T12:55:25.666212Z [HoneyPotSSHTransport,1,94.249.91.195] avatar root logging out
2025-05-23T12:55:25.666341Z [cowrie.ssh.transport.HoneyPotSSHTransport#info] connection lost
2025-05-23T12:55:25.666408Z [HoneyPotSSHTransport,1,94.249.91.195] Connection lost after 269.1 seconds
^C
```

Figure 31: Cowrie's Log Files of The Malicious Access

In the above figure it shows an IP address in the log files which can be confirmed to be the attacking machine's public IP address, ensuring correct logging of information.



What Is My IP?

YOUR CURRENT IP INFO	
My IPv4:	94.249.91.195
My IPv6:	2a01:9700:3563:4600:78ae:a65b:2ca0:3700
City:	Amman
State/Region:	Al 'Asimah
Postal Code:	-
Country:	Jordan
ISP:	Jordan Telecom Group
ASN:	8376
Time Zone:	UTC +03:00

Figure 32: Attacker's Machine Public IPv4 Address

5.1.2 Benign Hydra

To simulate a brute-force attack in a controlled and ethical manner, a custom Python script was developed using the Paramiko library. The script was designed to mimic tools like Hydra by attempting multiple SSH login attempts against Cowrie.

To ensure compliance with AWS's Acceptable Use Policy, the script was configured with a limited list of common passwords and included a delay between each login attempt to avoid triggering automated abuse detection systems.

```python
from paramiko import *
from time import *

target_ip = "16.171.0.195"
port = "2222"
username = "root"
passwords = ["123456", "admin", "password", "toor", "qwerty", "letmein", \
                "malak", "qabas", "cloud", "2025"]
delay = 1

print(f"[+] Starting brute-force simulation on {target_ip}...")

for password in passwords:
    print(f"[*] Trying password: {password}")
    ssh = SSHClient()

    try:
        ssh.set_missing_host_key_policy(AutoAddPolicy())
        ssh.connect(target_ip, port=port, username=username, password=
   password, timeout=5)
    except AuthenticationException:
        print(f"[-] Failed login with '{password}' (expected)")
    except Exception as e:
        print(f"[!] Error: {e}")
    finally:
        ssh.close()
        sleep(delay)

print("[+] Simulation complete!")
```

Listing 1: Benign Hydra

For each password in the predefined list, the script attempted to establish an SSH connection using Paramiko's SSHClient class. Each session was closed after the attempt, and a one-second delay was introduced between tries to simulate a low-intensity attack. This controlled simulation allowed for testing Cowrie's ability to log and detect brute-force attempts without violating AWS policies or generating excessive traffic.

Session	**Timestamp**	**Input**	**Source IP**
55092a39def7	2025-05-24T12:23:42.777067	root:2025	94.249.91.195
59591d2ba323	2025-05-24T12:23:41.153103	root:cloud	94.249.91.195
d55c43617d41	2025-05-24T12:23:39.521016	root:qabas	94.249.91.195
3d979a4e7f6c	2025-05-24T12:23:37.888752	root:malak	94.249.91.195
39e2c3ad9f1a	2025-05-24T12:23:36.224955	root:letmein	94.249.91.195
3e28e53b0d0c	2025-05-24T12:23:34.620922	root:qwerty	94.249.91.195
f3142eef06bb	2025-05-24T12:23:33.004842	root:toor	94.249.91.195
0163f62dae4a	2025-05-24T12:23:31.320914	root:password	94.249.91.195
8265b0ca4668	2025-05-24T12:23:29.676924	root:admin	94.249.91.195
d3dee1abb6a9	2025-05-24T12:23:27.016998	root:123456	94.249.91.195

The above table shows a subset of the log files that are parsed by the Lambda function and stored in DynamoDB, it clearly shows the username and password login attempts of the benign hydra python script as well as timestamps and IP addresses with each login attempt having a separate session ID. Cowrie was not affected or taken down by the repeated failed login attempts, demonstrating its robustness and stability under simulated attack conditions.

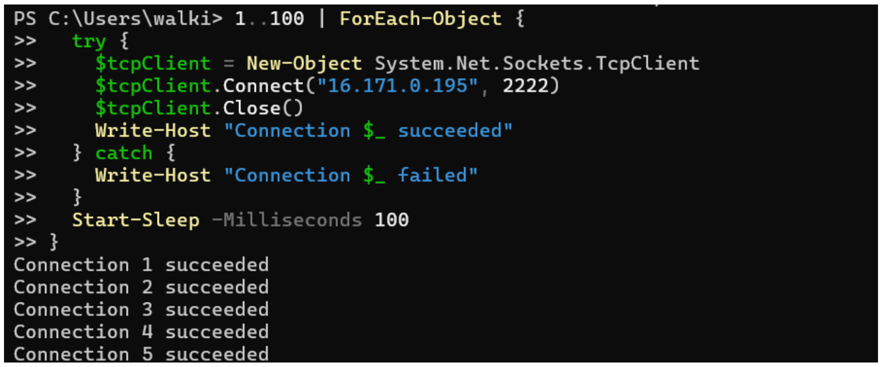
5.2 Performance and Scalability

5.2.1 PowerShell-Based TCP Flood Simulation (Consecutive)

To further evaluate the responsiveness and stability of the deployed Cowrie honeypot, a scripted test was conducted using PowerShell on a Windows host. The goal was to simulate repeated SSH connection attempts to the Cowrie service running on port 2222 of the EC2 instance.

```
1..100 | ForEach-Object {
  try {
    $ tcpClient = New-Object System.Net.Sockets.TcpClient
    $ tcpClient.Connect("16.171.0.195", 2222)
    $ tcpClient.Close()
    Write-Host "Connection $ _ succeeded"
  } catch {
    Write-Host "Connection $ _ failed"
  }
  Start-Sleep -Milliseconds 100
}
```

The script iterates 100 times, attempting to establish a new TCP connection to the honeypot on each iteration. A 100 ms delay is inserted between connections to control the rate and avoid overwhelming the client system. It was later scaled to iterate 500 times with a 1 ms delay and then 100 times with a 0.1 ms delay.
The purpose of the script is to simulate high-speed, repeated TCP connection attempts to test the honeypot's ability to handle rapid and concurrent access patterns, mimicking real-world scanner or botnet behavior.



```
PS C:\Users\walki> 1..100 | ForEach-Object {
>>   try {
>>     $tcpClient = New-Object System.Net.Sockets.TcpClient
>>     $tcpClient.Connect("16.171.0.195", 2222)
>>     $tcpClient.Close()
>>     Write-Host "Connection $_ succeeded"
>>   } catch {
>>     Write-Host "Connection $_ failed"
>>   }
>>   Start-Sleep -Milliseconds 100
>> }
Connection 1 succeeded
Connection 2 succeeded
Connection 3 succeeded
Connection 4 succeeded
Connection 5 succeeded
```

Figure 33: PowerShell TCP Flood Simulation

5.2.2 Python Asynchronous Flood Testing (Concurrent)

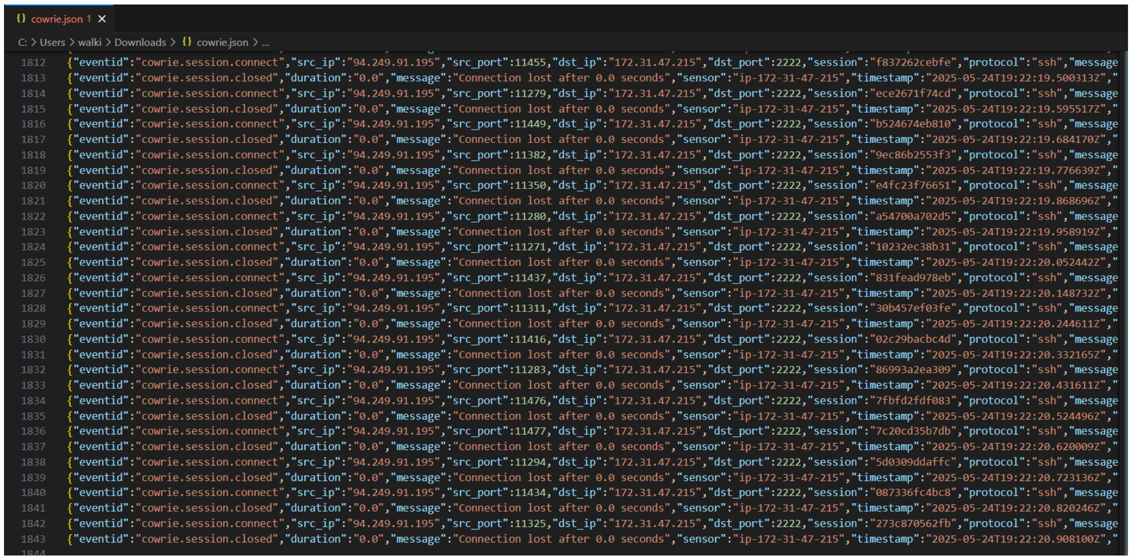
This Python script is designed to stress-test the honeypot by rapidly initiating 5,000 asynchronous TCP connection attempts to the target IP and port within a very short timeframe. Using Python's asyncio module, the script creates concurrent non-blocking tasks that simulate mass scanning or brute-force behaviors commonly seen in real-world attack scenarios.

```
import asyncio

async def flood(target_ip, target_port):
    try:
        reader, writer = await asyncio.open_connection(target_ip,
    target_port)
```

```python
        writer.close()
        print(f"Connected to {target_ip}:{target_port}")
    except Exception as e:
        pass

async def main():
    # EC2 instance's public IP
    target_ip = "16.171.0.195"
    target_port = 2222
    # 5k connections
    tasks = [flood(target_ip, target_port) for _ in range(5000)]
    await asyncio.gather(*tasks)

asyncio.run(main())
```

Listing 2: Async Flood Test

The objective is to evaluate the honeypot's resilience, scalability, and responsiveness under high-volume connection attempts, without overwhelming system resources. This test helps verify whether the honeypot can sustain realistic and aggressive traffic patterns while maintaining operational stability.



```
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11455,"dst_ip":"172.31.47.215","dst_port":2222,"session":"f837262cebfe","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:19.500313Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11279,"dst_ip":"172.31.47.215","dst_port":2222,"session":"ece2671f74cd","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:19.595517Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11449,"dst_ip":"172.31.47.215","dst_port":2222,"session":"b524674eb810","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:19.684170Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11382,"dst_ip":"172.31.47.215","dst_port":2222,"session":"9ec86b2553f3","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:19.776639Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11350,"dst_ip":"172.31.47.215","dst_port":2222,"session":"e4fc23f76651","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:19.868696Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11280,"dst_ip":"172.31.47.215","dst_port":2222,"session":"a54700a702d5","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:19.958919Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11271,"dst_ip":"172.31.47.215","dst_port":2222,"session":"10232ec38b31","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.052442Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11437,"dst_ip":"172.31.47.215","dst_port":2222,"session":"831fead978eb","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.148732Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11311,"dst_ip":"172.31.47.215","dst_port":2222,"session":"30b457ef03fe","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.244611Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11416,"dst_ip":"172.31.47.215","dst_port":2222,"session":"02c29bacbc4d","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.332165Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11283,"dst_ip":"172.31.47.215","dst_port":2222,"session":"86993a2ea309","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.431611Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11476,"dst_ip":"172.31.47.215","dst_port":2222,"session":"7fbfd2fdf083","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.524496Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11477,"dst_ip":"172.31.47.215","dst_port":2222,"session":"7c20cd35b7db","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.620009Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11294,"dst_ip":"172.31.47.215","dst_port":2222,"session":"5d0309ddaffc","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.723136Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11434,"dst_ip":"172.31.47.215","dst_port":2222,"session":"087336fc4bc8","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.820246Z","
{"eventid":"cowrie.session.connect","src_ip":"94.249.91.195","src_port":11325,"dst_ip":"172.31.47.215","dst_port":2222,"session":"273c870562fb","protocol":"ssh","message
{"eventid":"cowrie.session.closed","duration":"0.0","message":"Connection lost after 0.0 seconds","sensor":"ip-172-31-47-215","timestamp":"2025-05-24T19:22:20.908100Z","
```

Figure 34: Async Flood Test JSON Logs

The CPU utilization peaked at 69%, demonstrating that the instance had sufficient processing capacity to handle a high volume of simultaneous connection attempts without maxing out its resources.

This level of CPU efficiency indicates strong scalability headroom for more intense or prolonged attacks. From a networking perspective, metrics showed a sharp spike in inbound connections, confirming successful flood activity, yet no packet loss or throttling was observed. This suggests the underlying virtual network infrastructure effectively handled the burst load, and the honeypot remained operational and responsive throughout.

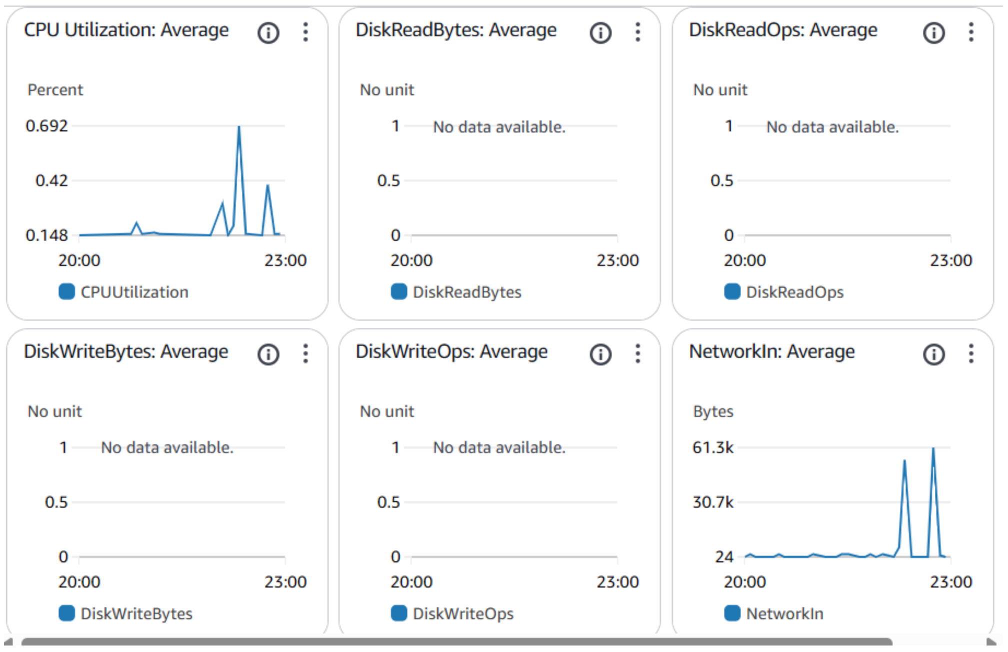


Figure 35: Resource Utilization during Tests

6 Challenges

Implementing a Honeypot-as-a-Service (HaaS) system introduces several technical, operational, and ethical challenges that must be carefully addressed to ensure the platform's effectiveness, scalability, and safety.

- **Security and Isolation**
  Since HaaS platforms intentionally attract malicious activity, strict isolation is crucial. Improper sandboxing may allow attackers to break out of the honeypot environment and access other systems or cloud services.

- **Cloud Infrastructure and Cost Management**
  Deploying and managing multiple honeypot instances in the cloud introduces complexity in orchestration, particularly when supporting multi-tenancy and dynamic scaling.

- **Scalability and Performance Bottlenecks**
  Handling large volumes of attack traffic while maintaining real-time responsiveness is a significant challenge.

- **Monitoring, Logging, and Forensics**
  Capturing sufficient data for analysis without overwhelming the system or alerting attackers requires a fine balance.

7 Conclusion

The development of a Honeypot-as-a-Service platform represents a proactive approach to cybersecurity, offering organizations valuable insight into attacker behavior and emerging threats. Using cloud infrastructure, the system has the potential to provide scalable, and customizable threat monitoring capabilities.

However, the project also introduces several critical challenges, including maintaining secure isolation, managing cloud resources efficiently, and navigating the ethical and legal complexities of handling real-world attack data.

Despite these challenges, a HaaS platform can significantly enhance an organization's defensive capabilities and contribute to broader research in threat intelligence. Through thoughtful implementation and continuous refinement, this system aims to strike a balance between realism, safety, and analytical depth—paving the way for safer digital environments.

References

[1] Amazon web services. Accessed: 2025-05-17. pages 5

[2] Cowrie (honeypot). Accessed: 2025-05-17. pages 4

[3] Cowrie honeypot documentation. Accessed: 2025-05-17. pages 6

[4] Honeypot. Accessed: 2025-05-17. pages 4

[5] What is cloud computing? Accessed: 2025-05-17. pages 4

[6] David French. How to setup “cowrie” — an ssh honeypot, 2023. Accessed: 2025-05-17. pages 4