Securing OpenStack based private IaaS Cloud environment With API Gateway and Honeypot

Aman Binod Singh
Dept. of Computer Science
Stevens Institute of Technology
Hoboken, NJ, USA
aman3singh.edu@gmail.com

Dr. Noor Ahmed
Dept. of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ, USA
nahmed9@stevens.edu

***Abstract*—Robust security measures become essential as private Infrastructure as a Service (IaaS) clouds continue to gain adoption in the market. This project involves the building and deployment of a dedicated private IaaS cloud environment with Openstack, a widely used open-source cloud platform. OpenStack relies on Application Programming Interface(API) for performing and implementing some of the most core cloud operations. As OpenStack uses TLS for securing API communication, these APIs have become a frequent target for malicious users who exploit such vulnerabilities to inject malicious payloads into the request going to OpenStack services. To secure Openstack cloud environment against these known API vulnerabilities, we first built a private cloud environment and deployed it using cloud native tools, and then implemented a NGNIX based API gateway with a dedicated Honeypot service. This API gateway acts as the first layer of defense, routing, and filtering requests, while the honeypot mimics legitimate OpenStack API endpoints to capture and analyze malicious activity, enabling comprehensive threat intelligence gathering and attack profiling. Preliminary results demonstrate the capability of this architecture to effectively detect, analyze, and mitigate API-based threats.**

***Index Terms*—OpenStack, Honeypot, API Gateway, Kolla-Ansible, Docker**

I. Introduction

With the ever increasing reliance on private Infrastructure as a Service (IaaS) cloud within the private and government organizations, the need for robust measures ensuring high availability and security for these cloud environments is paramount. OpenStack is a widely used open source cloud computing platform for deploying IaaS cloud solutions. OpenStack leverages APIs to execute essential operations, including user authentication and authorization, managing networking services that interconnect core OpenStack components, and most crucially, facilitating the creation and lifecycle management of virtual machines. However, these APIs have become attractive targets for malicious actors, with common exploits such as token theft, replay attacks, SQL injection, often used to compromise cloud operations [1].

This project focuses on developing a private IaaS cloud with OpenStack and deploying it using cloud native automation scripts such as Kolla-Ansible [2]. While Kolla-Ansible automates the deployment of core OpenStack services, we have also integrated the OpenStack services Ceilometer and Senlin to ensure high availability and automated failure recovery within the cloud environment.

We have then implemented a NGNIX based API gateway with a dedicated honeypot service responsible for detecting, capturing, and logging malicious API-based attacks, serving as the primary security mechanism for the platform. This approach combines traditional security measures with advanced deception methodologies, creating a multi-layered defense system. By deploying a dedicated API Honeypot which mimics OpenStack API endpoints, we can observe and mointor the attackers pattern and behaviour, collect valuable threat intelligence, and develop adaptive countermeasures.

The objective of this private cloud is to deploy all ECE 800 ML related projects for Center for Innovative Computing and Networked Systems(iCNS) at Stevens Institute of Technology.

The remainder of the paper is organized into four sections. Section II formally defines the research problem, highlighting the key problems and shortcomings we are trying to address and solve. Section III details the deployment of the private IaaS cloud using OpenStack, along with the architecture and implementation of the NGNIX-based API Gateway and the Honeypot. In Section IV we analyze the outcomes from our API gateway and honeypot service to assess the effectiveness of our security solution within the OpenStack cloud environment. And lastly in Section V, we present a concise conclusion of the project and explore the future scope related to our work.

II. Problem Statement

This project involves deploying a private IaaS cloud environment using Openstack (specifically Openstack Version ZED). Some of the core components of Openstack which we have deployed include Nova, Keystone, Neutron, Glance, Swift and Cinder - all of them relying on API requests to facilitate critical tasks of creating compute instances and their life-cycle management, authentication and networking between the core components facilitated by Neutron API. OpenStack APIs are typically secured using token-based authentication mechanisms and TLS encryption [1] but these security measures remain vulnerable to a variety of sophisti-

cated threats. Specifically, this project addresses the following critical security issues associated with OpenStack APIs:

- API Request Tampering and Payload Injection Attacks: Even with the implementation of TLS, APIs remain vulnerable to injection attacks like SQL injections and command injections. Such requests with malicious payload can compromise the data integrity and affect cloud operations, if not systematically dealt with [1].
- Inadequate Rate-Limiting and Denial-of-Service(DoS) attacks: Openstack security policies do not enforce proper rate-limiting of API requests coming to the various core components. Poorly implemented rate-limiting policies can expose the entire cloud environment to DoS attacks, potentially affecting system resources and availability [1].
- Lack of Comprehensive Threat Monitoring: Rule-based detection systems are based on predefined rules that are below par against unknown or adaptive threats like zero-day attacks.
- Limited Threat Intelligence Gathering: Current OpenStack deployments typically spend their efforts blocking and dealing with known threats rather than understanding the attack methodologies and motivations, which limits the ability of the system to preempt future threats.

By generating detailed logs of suspicious requests, the honeypot will enable administrators and security systems to identify sophisticated or previously unknown attacks that would otherwise evade detection by static, rule-based security measures. NGNIX based API gateway will provide a solid defense mechanism - reinforcing the entire cloud environment with a multi-layered security approach.

III. Solution

In this project, we have deployed a production-grade Openstack based Private IaaS cloud by making use of Kolla-Ansible automation scripts. Kolla-Ansible provides containerized OpenStack services that are ready to be deployed, making it a highly robust and scalable cloud deployment strategy.

The cloud environment was built and deployed on three dedicated virtual machines (VMs) hosted on a local Fedora machine. These VMs which are named as follows: infra-node, control-node, and compute-node, run Rocky Linux 9.5 as their operating system, with Kolla-Ansible configured on the infra-node [3]. Detailed descriptions of these VMs are provided later in subsection "Private Iaas Cloud Implementation" of the Solutions section. Each node is configured with sufficient memory, vCPUs, and network interfaces to support the deployment and operation of core OpenStack services, including Ceilometer and Senlin, to ensure high availability. This architecture replicates a production-grade cloud environment by utilizing dedicated VMs and deploying essential OpenStack services for reliable cloud functionality.

A. System Architecture

Illustrated in Fig. 1 is the system architecture of the Private IaaS cloud deployment using VM's with the NGNIX API gateway as the only entry point for the API requests coming to Openstack and a dedicated Honeypot service

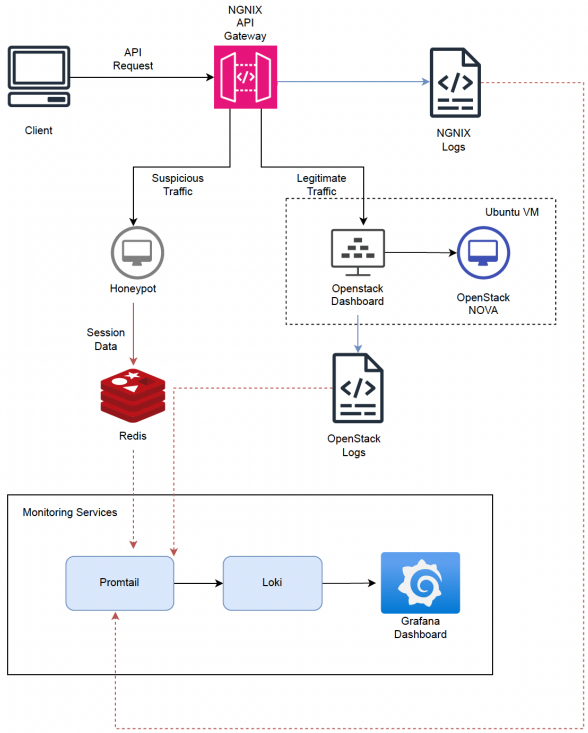

Fig. 1. *System Architecture*

1) Private Iaas OpenStack Cloud Environment: Deployed using Kolla Ansible on three VMs (infra-node, control-node, and compute-node) running Rocky Linux 9.5. This deployment hosts essential OpenStack components accessible via APIs, including Nova (compute service), Neutron (network service), Keystone (identity service), and Glance (image service). This robust and distributed architecture supports high availability, scalability, and optimized resource allocation for compute instances.
2) API Gateway: An NGNIX-based API gateway acts as the single point of entry for all requests routed to the OpenStack services. Based on predefined rules, responsible for load-balancing, rate-limiting, routing incoming traffic to either legitimate OpenStack-services or the honeypot.
3) Honeypot Service: Flask-based application that mimics the OpenStack Nova service (component responsible for compute resource management), deployed as a Docker container. Responsible for capturing suspicious and malicious requests and providing dynamic responses to such suspicious requests [5].
4) Redis Database: Deployed as a Docker container, to

store session data, attack patterns ,and behavioral information collected from potential attackers, making use of Redis's high-performance data storage capabilities.

5) Monitoring and Logging Services: A centralized log collection and data visualization service designed to showcase analytics and evaluate the effectiveness of the API Gateway and Honeypot service. Consists of Promtail, Loki and Grafana , where all services are deployed as Docker containers.

B. Network Flow and Request Handling

The system processes incoming API requests as as illustrated in Fig. 2

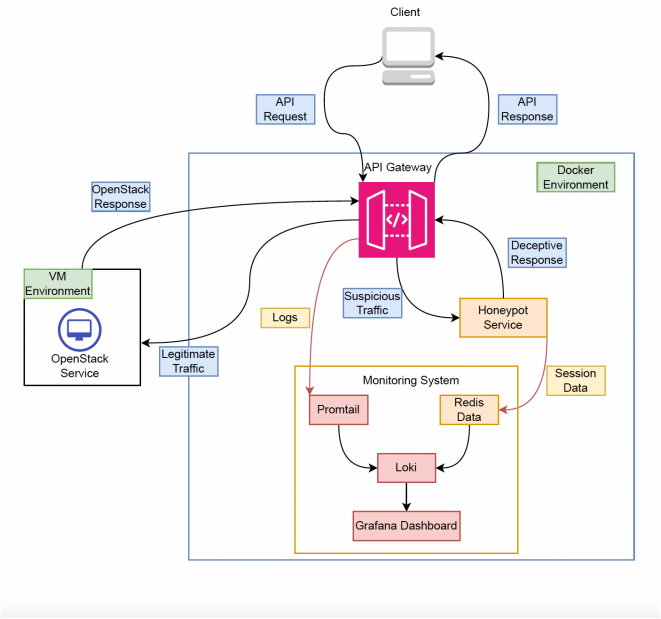

Fig. 2. *API Request/Response information flow*

1) All requests arrive at the NGNIX-based API gateway, which is listening on port 8080, serving as the single entry point for all of the request directed for the system.
2) The gateway then examines every request's certain characteristic (which includes source IP address, request rate, HTTP headers, request payload) to make the routing decision against pre-defined rules.
3) Legitimate API requests are routed to the appropriate OpenStack services based on their intended functionality.
4) Suspicious requests are redirected to the honeypot which mimics the OpenStack services. It also sends dynamic response to the attacker - making the attacker think that it is interacting with a legitimate OpenStack service [6].
5) The honeypot uniquely identifies user based on IP address and user agent, allowing the system to build a behavior profile of suspicious user and their patterns. This data is then sent to Redis database, which helps researcher with real-time analysis as well as post-attack forensic investigation [7].
6) Promtail collects system logs from various services and forwards them to Loki for storage. Loki processes the log data in JSON format and sends it to the Grafana dashboard, where it is visualized for analysis.

C. Private Iaas Cloud Implementation

1) Proof Of Work: Initially, to demonstrate the feasibility and foundational understanding required for our project, we built a prototype private IaaS cloud using DevStack [4]. DevStack is an open-source collection of scripts designed for simplified OpenStack deployments, particularly suitable for development and testing purposes. This prototype deployment was implemented on an Ubuntu Virtual Machine configured with 8 GB of RAM and 60 GB of storage, providing adequate resources for the smooth operation of core OpenStack services.

Using this DevStack installation, we successfully configured and tested the core OpenStack components of:

- Horizon: Openstack's Dashboard that enables users to log in and put requests for various operations.
- Nova (Compute): Responsible for managing the lifecycle of virtual machine instances.
- Keystone (Identity): Provides authentication and authorization services for users and API requests.
- Glance (Image): Manages disk and server images.
- Neutron (Networking): Handles networking services for virtual machines.
- Cinder: The block storage service for Openstack which provides persistent storage for compute VM instances and VM snapshots.

This preliminary prototype cloud, developed using DevStack, enabled us to understand and replicate a production-grade OpenStack deployment with the essential core services, allowing us to establish a functional testing environment with significantly fewer computational resources. It provided a practical foundation for the subsequent development, testing, and evaluation of our security architecture, specifically the API gateway and Honeypot systems.

2) Production-Grade Deployment with Kolla-Ansible: Following the successful prototype cloud environment deployment using DevStack, we transitioned to a robust, scalable, and maintainable production-level OpenStack deployment using Kolla-Ansible. This production-grade cloud implementation involved creating three dedicated virtual machines (VMs): infra-node, control-node, and compute-node, all running Rocky Linux 9.5 as illustrated in Fig. 3

- infra-node: This node serves as the orchestration point for deployment. Using Kolla-Ansible scripts, it manages the deployment, configuration, and updating of OpenStack services across the entire infrastructure [3] [4].
- control-node: Central node which hosts critical OpenStack management services, including Keystone which is

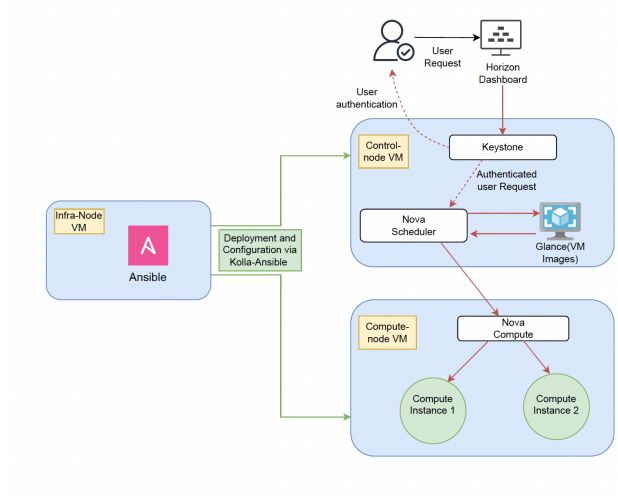

Fig. 3. *OpenStack platform building blocks*

used for Identity and Access Management, Glance which is the central directory for handling and maintaining VM images , Horizon dashboard , Nova Scheduler which decides how to dispatch resources for compute instance requests, Neutron which handles the entire internal networking between the OpenStack services, and Cinder which provides the block storage for OpenStack cloud environment.

- compute-node: A node equipped with GPU passthrough capability from the host Fedora machine, dedicated to hosting and running virtual machines. This node supports GPU-intensive machine learning workloads running on compute instances requested by users. The node effectively manages the compute instances through OpenStack's Nova Compute, where instance creation requests and resource allocation instructions are sent by the Nova Scheduler service.

The high-level process of creating compute instances in the OpenStack cloud environment is as follows: A user accesses the OpenStack cloud via the Horizon Dashboard, where the Keystone service authenticates the user and assigns predefined access controls. Once authorized, the user submits a compute instance request, which is validated by Keystone and then forwarded to the Nova Scheduler on the control node.

The Nova Scheduler is responsible for selecting the most suitable compute node for instance creation, based on factors such as resource availability, current workload, and configured scheduling policies. It evaluates the request and generates an API call to the Nova Compute service running on the compute node. This API call includes all necessary details specified by the user, including instance configuration and resource requirements.

During this process, the Nova Scheduler also interacts with the Glance service to retrieve the required VM image necessary for instance creation. Once the API call is prepared, it is transmitted to the Nova Compute service over RabbitMQ , which serves as the messaging broker between OpenStack components - as illustrated in Fig. 3.

Upon receiving the API request, the Nova Compute service running in the compute-node, provisions and launches the compute instance, finalizing the process and making the new instance available to the user.

The Neutron service assigns floating IP addresses to compute instances launched by the compute node. It also sets up the instance's network interface within a network bridge linked to the compute-node VM's interface, enabling external internet connectivity. [2].

This structured deployment enabled us to create a stable and high-performance environment suitable for hosting advanced workloads. It significantly enhanced the reliability and scalability required to implement and rigorously evaluate our security mechanisms, such as the API Gateway and Honeypot systems. Furthermore, choosing Rocky Linux 9.5 as the operating system for the nodes ensured long-term stability, security, and compatibility with the Kolla-Ansible deployment method.

D. Honeypot Implementation

The Honeypot service functions as the fundamental element in addressing various key-problems that we want to solve with OpenStack's deployment of a private IaaS cloud environment. Implemented in Python using the Flask framework, the honeypot provides the following key functionalities:

1) Session Tracking and Attacker Behavior Profiling with Redis integration: The honeypot identifies and monitors potential attackers across multiple requests by implementing session tracking, where each user is uniquely identified using its source IP address and the user agent present in the HTTP header of the API request. This helps the system build a comprehensive behavior profile of the attacker and their attack patterns [7].

All of the session data is stored in Redis database, which is deployed as a Docker container. This helps retain the session data even if the containers restart, update or fail. In Redis, the system maintains malicious users request history, which consists of all the different types of requests made by the user, the endpoints accessed and the interaction depth.

The system tracks five distinct levels of interaction depths, where each level represents increased access to the sensitive system information. This is similar to setting up access for the user based on the Least Privilege Access Principle.

- Level 1: Basic authentication and token acquisition from OpenStack's IAM service KeyStone
- Level 2: Read-Only access to send GET request to various OpenStack API endpoints
- Level 3: More detailed Read-Only access to do resource inspection

- Level 4: Access to create and start VM using the NOVA compute
- Level 5: Administrative access

This hierarchical segregation of attacker interaction levels enables us to analyze and understand attackers' patterns and strategies for penetrating the cloud environment—an aspect currently lacking in the existing OpenStack security architecture.

2) User Authentication Simulation: Manipulating and misusing stolen or proxy authentication tokens is a prominent strategy used by attackers to gain unauthorized access in the cloud environment. To understand how attackers exploit authentication tokens and to create a behavior profile of the attacker, the honeypot generates and assigns fake tokens to the attacker. The attacker uses this fake token to access various services mimicked within the honeypot. The system tracks these fake tokens to monitor API requests made by the attacker, recording both the time of the first request and the most recent interaction with the honeypot system [6]. All of this session data of the attacker is stored in the Redis database.

E. Logging and Monitoring Infrastructure

A central feature of our system is a production-grade monitoring and alerting setup that provides real-time visualization of threats targeting the cloud environment, enabling us to take quick responses against threat when anomalies are detected on the dashboard.

Log Collection and Storage: As the API gateway and the Flask-based Honeypot service are deployed as a Docker container, we can easily generate and collect system logs for these services. We make use of Grafana and its services to collect and visualize logs [4].

- Promtail: Acts as the log collection agent, scraping logs from all Docker containers, processing raw data to extract useful information, and forwarding it to Loki.
- Loki: One of the Grafana components which is used for log aggregation and the backend log storage system for Grafana. It implements an efficient indexing system, where it indexes the metadata(labels) of the log rather than the entire log, enabling fast execution of queries and analysis.
- Grafana Dashboard: Grafana uses processed logs stored in Loki to visualize data on its dashboard. Custom dashboards can be created to monitor key security metrics and set alerts for anomalies in real-time data [8].

As illustrated in Fig. 4, we have created our own custom Grafana dashboard to visualize and demonstrate the effectiveness of our API gateway and Honeypot services. The dashboard shows the following key security metrics:

- API Request Rate: The dashboard visualizes the frequency of API requests that are incoming at the API gateway, enabling the users to see any unusual spikes in the incoming traffic.
- Route Decisions: A pie chart visualizing the API gateway's routing decisions, helping distinguish between legitimate and malicious incoming traffic.
- Status Code Distribution: A bar gauge displaying HTTP response status codes from the gateway that are being sent from the various services, enabling quick identification of irregular patterns beyond status code 200.
- Top Client IPs: A table displaying the IP addresses with the highest request volume, helping identify persistent attackers or bots potentially launching DoS attacks.

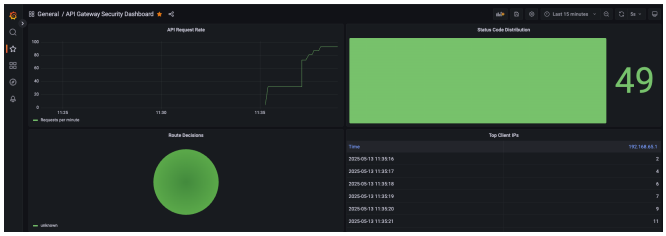

Fig. 4. *Grafana Dashboard*

These visualizations demonstrate the effectiveness of the API gateway and Honeypot service, highlighting the system's ability to actively monitor and respond to threats.

IV. Experimental Evaluation and Results

To test out our API Gateway and how it is handling the various API requests, we wrote various Python scripts which simulate different kinds of attack and send different kind of requests to the gateway. Upon sending the request, we look at the NGNIX gateway logs and the responses received for the request to evaluate how efficiently the gateway and the Honeypot service are working.

A. Testing for suspicious header in the incoming API request

To enforce stricter security rules, we configured the NGINX gateway to inspect the headers of incoming API requests. If the header contains suspicious user agents, such as those from tools like Nmap or Nikto, or indicates the use of a proxy service that masks user identity, the API gateway will automatically redirect these requests to the Honeypot service.

We used the same Python testing script from the rate-limiting test to evaluate various API requests containing suspicious, proxy, and AWS service user agents in the request headers. As shown in Fig. 5, when the request contains suspicious or proxy headers, it gets redirected to the Honeypot, returning a status 200 response. Although the response indicates routing to OpenStack, the service appears as "unknown" to make the user believe they have reached the OpenStack endpoint. However, in Fig. 6, when using AWS headers, the response shows a status 200 from the "openstack-mock" service, which acts as a testing endpoint for OpenStack API health.

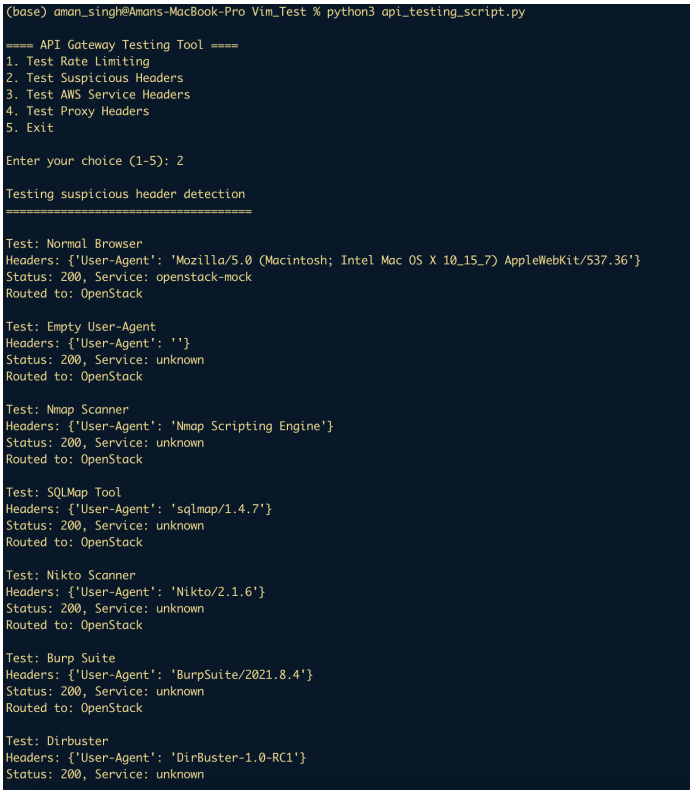

Fig. 5. *Testing API requests with suspicious user-agents in the request header*

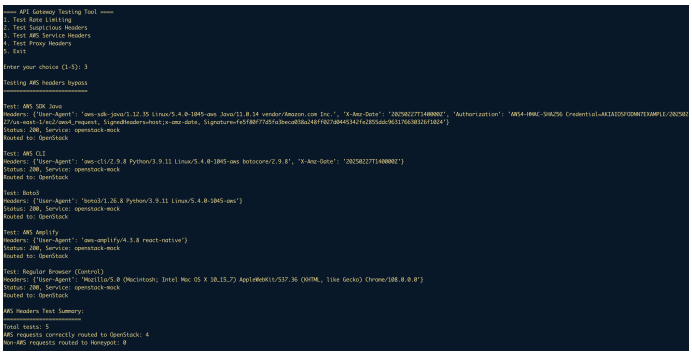

Fig. 6. *Testing API requests with AWS user-agents in the request header*

B. Testing API Rate-Limiting

OpenStack lacks effective rate-limiting for API requests, making it vulnerable to DoS attacks when overwhelmed with high volumes of requests. To address this, we implemented a rate-limiting protocol in the NGINX API Gateway, restricting users to 10 requests per second. If the gateway detects a burst exceeding this limit, it flags the activity as suspicious and redirects the traffic to the Honeypot service.

As illustrated in Fig. 7 and 8 we tested the API Gateway by running a Python script on our local-host, which rapidly sent 30 API requests to OpenStack. Initially, the script checked the availability of OpenStack services by executing a ”curl -v http://localhost:8080/health” command, verifying that the control-node VM was accessible on port 8080. Once confirmed, the script sent a burst of 30 same health requests through the gateway.

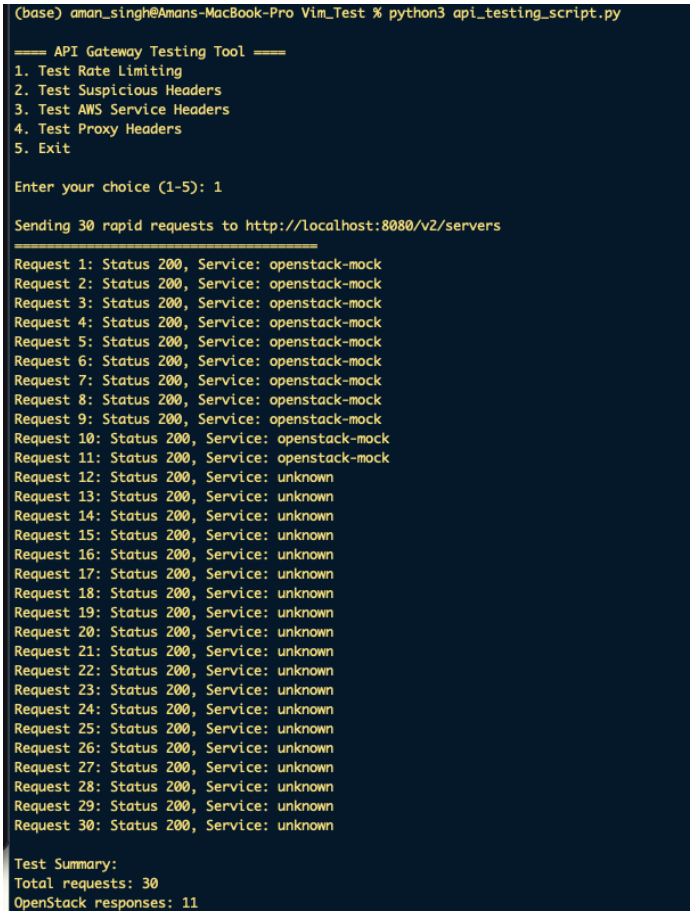

Fig. 7. *Testing Rate-Limiting using python script*

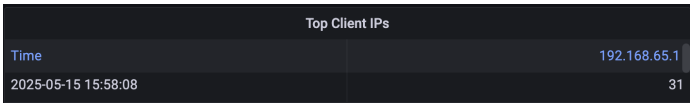

Fig. 8. *Dashboard displaying top clients and their IPs by API request volume*

Up to the 10th request, OpenStack responded with a status 200. However, when the 11th request hit the NGINX gateway, the rate-limiting rule was triggered, redirecting subsequent requests from the same IP (localhost in our case) to the Honeypot service. From the 12th request onward, we received a status 200 from an ”unknown” service, indicating that the Honeypot was handling the requests. Further checking the Grafana dashboard during the test, as illustrated in Fig. 9 and 10, we observed a spike in the API request rate at the gateway. The dashboard also highlighted the IP address with the highest request volume and displayed the response status codes sent out from the NGINX gateway to the user, all of which were status 200 in our case.

Fig. 9. *Dashboard showcasing API requests per minute to the gateway*

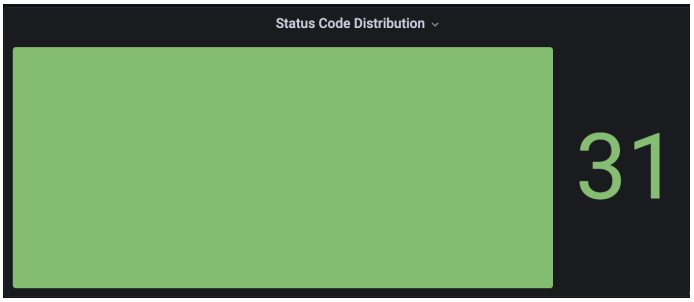

Fig. 10. *Dashboard showcasing status code in the response going out from the gateway*

C. Profiling attacker behavior

As described in the ”Honeypot Implementation” subsection, the honeypot service simulates a fake authentication process by assigning a fake token to track attacker behavior. To demonstrate this, we created a Python script that mimics an Nmap request by setting the user-agent to ”nmap.” The NGINX gateway redirects this request to the Honeypot, which then assigns a fake authentication token, giving the impression that the attacker has access to OpenStack services, as shown in Fig. 11, mimicking Keystone service of OpenStack.

```
(base) aman_singh@Amans-MacBook-Pro Testing_Scripts % python3 ./honeypot_test.py --mode sequential --show-response --agent "nmap/1.0"

POST http://localhost:8080/v2/auth/tokens
Status: 200
Response:
{
  "token": {
    "expires_at": "2025-05-16T23:09:03.872967",
    "id": "f8ac43d8-8d7d-4b88-8917-a6943e96745b",
    "issued_at": "2025-05-15T23:09:03.872924",
    "tenant": {
      "description": "Default tenant",
      "enabled": true,
      "id": "default-tenant-id",
      "name": "default"
    }
  }
}
Received token: f8ac43d8-8d7d-4b88-8917-a6943e96745b
```

Fig. 11. *Honeypot assigning fake authentication to attacker*

After receiving the auth-token, the script uses it to access the Honeypot which mimics various OpenStack core services. The Honeypot then creates an attacker behavior profile by tracking all interactions made using the fake token, including the accessed endpoints, the interaction level reached, and the timestamps of the first and last observed activity, as illustrated in Fig. 12. It also saves the data of all requests sent by the attacker to the various services. The attacker session data are then stored in a Redis database for future use and for generating security reports.

```
(base) aman_singh@Amans-MacBook-Pro Vim_Test % curl http://localhost:8775/sessions
{
  "172.18.0.6:nmap/1.0": {
    "endpoints_accessed": [
      "GET:/v2/servers/server-1",
      "GET:/v2/volumes/volume-3",
      "GET:/v2/admin/config",
      "GET:/v2/volumes",
      "GET:/v2/images",
      "GET:/v2/servers",
      "POST:/v2/auth/tokens"
    ],
    "fake_token": "483c59f4-1d26-4df5-8636-1ed0364f9bea",
    "first_seen": 1747349743.320198,
    "interaction_level": 5,
    "ip": "172.18.0.6",
    "last_seen": 1747349749.4216504,
```

Fig. 12. *Attacker behavior profiling*

V. Conclusion

With the successful production-level deployment of our private IaaS cloud environment using OpenStack and Kolla-Ansible, integrated with a comprehensive security framework consisting of an NGINX-based API gateway and a Flask-based honeypot system, we have demonstrated substantial improvements in detecting, profiling, and mitigating API-based threats.

Future work can focus on integrating machine learning models to detect new and unknown threats by analyzing extensive logs collected by Promtail and attacker behavior profiles stored in Redis. These models will enable real-time API traffic analysis, classify malicious requests, and identify zero-day attack patterns. Additionally, enhancing the Grafana dashboard to visualize security-specific metrics and generate custom reports would improve monitoring and analysis.

Acknowledgment

The author would like to thank Dr. Norman Ahmed, for his guidance and support throughout this project.

References

[1] J. Yu, L. Fu, P. Liang, A. Tahir and M. Shahin, ”Security Defect Detection via Code Review: A Study of the OpenStack and Qt Communities,” 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), New Orleans, LA, USA, 2023, pp. 1-12, doi: 10.1109/ESEM56168.2023.10304852.

[2] Omar Khedher, Mastering OpenStack: Implement the latest techniques for designing and deploying an operational, production-ready private cloud , Packt Publishing, 2024.

[3] H. T. Ciptaningtyas, R. R. Hariadi, M. Husni, K. Ghozali, R. W. Sholikah and I. M. D. Setyadharma, "OpenStack Implementation using Multinode Deployment Method for Private Cloud Computing Infrastructure," 2023 International Seminar on Intelligent Technology and Its Applications (ISITIA), Surabaya, Indonesia, 2023, pp. 12-17, doi: 10.1109/ISITIA59021.2023.10221042.

[4] M. Moravcik and P. Segec, "Automated deployment of the OpenStack platform," 2023 21st International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 2023, pp. 375-380, doi: 10.1109/ICETA61311.2023.10343784.

[5] J. Buzzio-Garcia, "Creation of a High-Interaction Honeypot System based-on Docker containers," 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4), London, United Kingdom, 2021, pp. 146-151, doi: 10.1109/WorldS451998.2021.9514022.

[6] U. Bartwal, S. Mukhopadhyay, R. Negi and S. Shukla, "Security Orchestration, Automation, and Response Engine for Deployment of Behavioural Honeypots," 2022 IEEE Conference on Dependable and Secure Computing (DSC), Edinburgh, United Kingdom, 2022, pp. 1-8, doi: 10.1109/DSC54232.2022.9888808.

[7] Á. Balogh, M. Érsok, A. Bánáti and L. Erdődi, "Concept for real time attacker profiling with honeypots, by skill based attacker maturity model," 2024 IEEE 22nd World Symposium on Applied Machine Intelligence and Informatics (SAMI), Stará Lesná, Slovakia, 2024, pp. 000175-000180, doi: 10.1109/SAMI60510.2024.10432876.

[8] A. Mehdi, M. K. Bali, S. I. Abbas and M. Singh, ""Unleashing the Potential of Grafana: A Comprehensive Study on Real-Time Monitoring and Visualization"," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-8, doi: 10.1109/ICCCNT56998.2023.10306699.