Stinger: Network Scanner as a Service

Scanner module for Intelligent Software Defined Security System

Shantanu Bhusari
School of Computing Informatics and Decision Systems Engineering
sbhusari@asu.edu

Chandramouli Janakiraman
School of Computing Informatics and Decision Systems Engineering
cjanakir@asu.edu

Raghuveer Gowda H
School of Computing Informatics and Decision Systems Engineering
raghuvee@asu.edu

***Abstract* – Though SDN provides us with key features such as programmability and virtualization for cloud systems; security, usability and availability remains critically important in real-life deployment. A cloud system becomes easily vulnerable to attacks if necessary precautions are not taken, as it usually allows users to install any applications or data on it. Intelligent software defined security system (ISDS) provides that security solution for cloud systems by keeping track of running services and possible vulnerabilities. This project focuses on creating an automated module that will gather such information including open ports, detailed information of running services on those ports, possible vulnerabilities, CPU utilization, network usage and the diagnostics data of VM in the given cloud topology. Periodic scanning will be performed in the given topology and information will be stored in the database. This information will be provided to the other modules of ISDS through RESTful APIs.**

Keywords - ISDS, OpenStack, OpenVAS, Nmap, Nova API, MongoDB, Python Flask, Restful APIs

I. Introduction

a. Following problems are addressed in our project:

1. Cloud systems are easily vulnerable to security issues as the user has control to install vulnerable applications, which may give the attacker an option to deny any of our services or gain access to our system using remote code execution.

2. An even worse scenario would be when the compromised machine is used to access other systems on our network.

b. Why these problems are important?

Security is the major concern in current internet standards. There are various vulnerabilities which possesses threat to the complex structure of cloud system. Any misconfiguration or malicious behavior can affect a huge number of systems and users on that cloud network.

Availability and Usability are other important factors in any user-friendly application. It is crucial for a cloud system to handle given tasks in efficient way without compromising the availability and usability. Prediction of possible threats can help in taking necessary precautions to maintain the system running with full efficiency. Thus, finding the vulnerabilities and possible attack scenarios beforehand becomes critical in an intelligent security system.

c. Applied technologies and solutions to address these problems.

To develop this module, we are using OpenStack to create the virtual topology of cloud network. We will be using Nmap wrapper and OpenVAS scanner to scan for available/running services, open ports and network vulnerabilities. Network Monitoring will be performed using Nova API tool. The information will be stored in a database using MongoDB.

d. Expected outcomes

Expected outcome is providing automated network scanning as a service which will be integrated with other modules. This scanner will provide information through REST APIs which will be used by other modules. This module will also abstract the ad hoc scan by performing regular periodic scans and providing the necessary information from the database when an ad hoc scan query is executed by the user, thus enhancing the availability of the module.

e. Project management plan (timeline, and group members, etc.)

The duration of the project is two months which includes development of the system, testing, documentation and reporting of the same, which will be done by Shantanu Bhusari, Chandramouli Janakiraman and Raghuveer Gowda H.

II. System Models

A. System Model

The system we are developing consists of a set of virtual machines which will form a network topology. OpenStack will be used to create this cloud network topology. Nova APIs provided by OpenStack will be used to gain the topology information periodically. Various scripts will be created to fetch the required information of running services on open ports, possible vulnerabilities in the network, as well as to check the health of each VM for CPU usage and network traffic. This information will be stored in a database in JSON format and will be exposed through REST API's whenever queried. The entire process will be automated, and better usability will be achieved by running the scripts as a background process without interrupting any other running services.

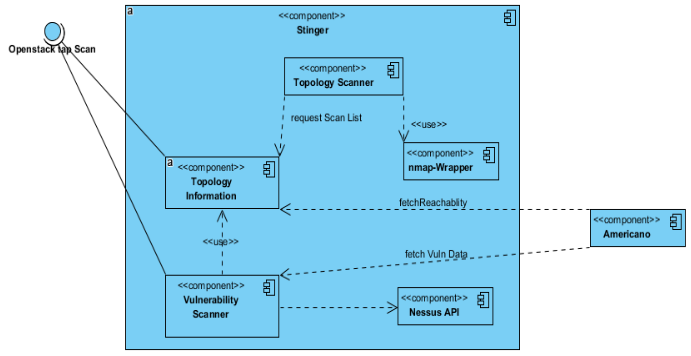


Fig1 Network Scanner use case diagram

B. Software

OpenStack (DevStack), OpenVAS, Nmap, OpenStack APIs, OpenStack SDK, Python, Flask, and MongoDB.

C. Threat Model

For this project, we are assuming that the hypervisor is trusted and secured. It means that the hypervisor isolates the resources and environment for the VMs. The model also assumes that users have capabilities to install and execute any software on VMs. We are also considering the possibility of inside threat, which means a compromised system could be present inside the network. Network Administrator will have control of the OpenStack and the scanners to find the possible vulnerability information.

Use Case Diagram:

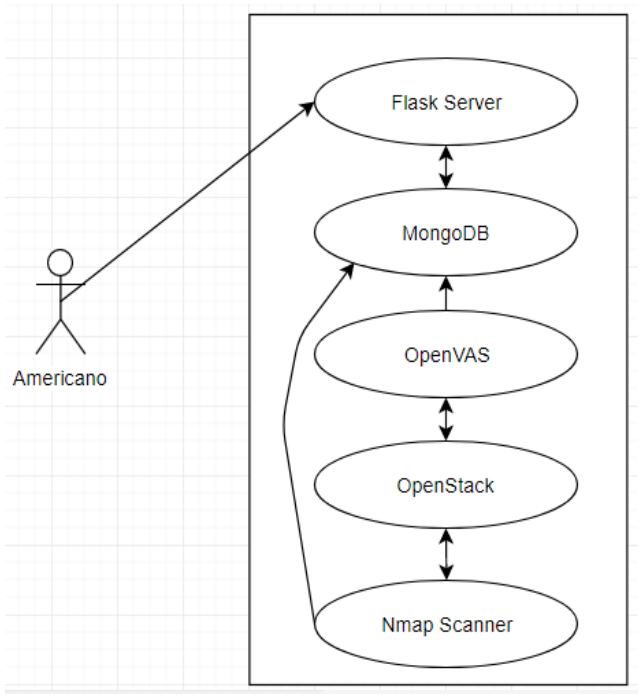
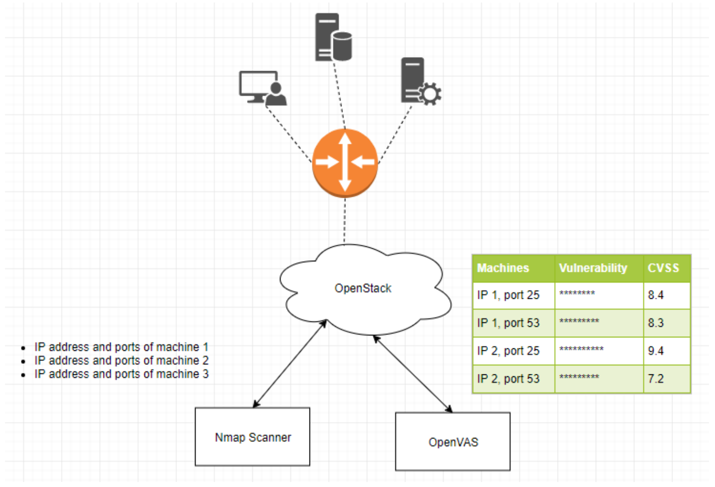


Fig2 Project Use case diagram

- Initially the Americano module will request for the vulnerability information from the stinger module by invoking the flask server.
- The flask server internally fetches the information from MongoDB which has been stored using the previous Nmap and OpenVAS scan.
- Nmap scanner will scan the OpenStack network and stores the running services and open port information to the database periodically.
- OpenVAS will scan the virtual machine present inside OpenStack for the existing vulnerabilities and stores it in database periodically.

III. Project Description

The aim of this project is to provide API's for fetching the required information. Users from other modules will be able to make queries for a range of IP addresses and port numbers, thus making this module compatible to different combination of queries. Automation will be used in achieving regular updates by storing the scanned results in the database, so that the results can be fetched by other modules like Americano for attack graph generation whenever required.


OpenStack
IP address and ports of machine 1
IP address and ports of machine 2
IP address and ports of machine 3
Nmap Scanner
OpenVAS

Machines	Vulnerability	CVSS
IP 1, port 25	********	8.4
IP 1, port 53	*********	8.3
IP 2, port 25	**********	9.4
IP 2, port 53	*********	7.2

Fig3 Topology of Core Functionality

A. *Project Overview*

This project consists of the following tasks.

Task 1: Setup OpenStack with latest version

DevStack is an opinionated script to quickly create an OpenStack development environment. It can also be used to demonstrate starting/running OpenStack services and provide examples of using them from a command line. So, we are making use DevStack which is development version of OpenStack.

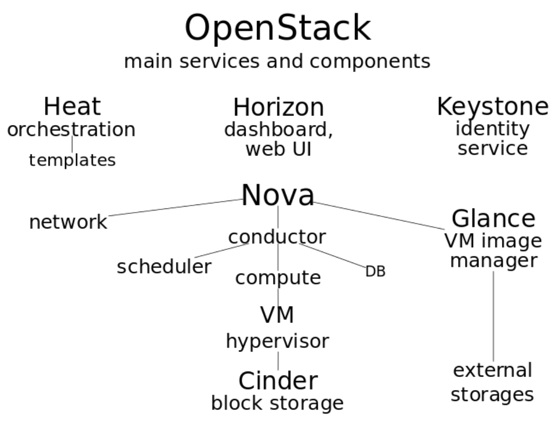


Fig4 OpenStack Core Components

<u>Compute (Nova)</u>
OpenStack Compute (Nova) is a cloud computing fabric controller, which is the main part of an IaaS system. It is designed to manage and automate pools of computer resources and can work with widely available virtualization technologies.

<u>Networking (Neutron)</u>
OpenStack Networking (Neutron) is a system for managing networks and IP addresses. OpenStack Networking ensures the network is not a bottleneck or limiting factor in a cloud deployment, and gives users self-service ability, even over network configurations.

<u>Identity (Keystone)</u>
OpenStack Identity (Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP.

<u>Object storage (Swift)</u>
OpenStack Object Storage (Swift) is a scalable redundant storage system. Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster.

<u>Dashboard (Horizon)</u>
OpenStack Dashboard (Horizon) provides administrators and users with a graphical interface to access, provision, and automate deployment of cloud-based resources.

DevStack is configured to create and handle the network topology. Network Topology created in our setup is as shown in the below figure. Two networks NET1 and NET2 are connected to Internet via public gateway as shown in the below figure. Both NET1 and NET2 is configured with Metasploitable VMs which have many vulnerable services for testing purpose.

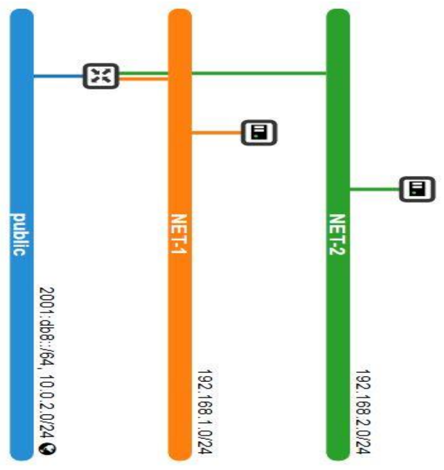
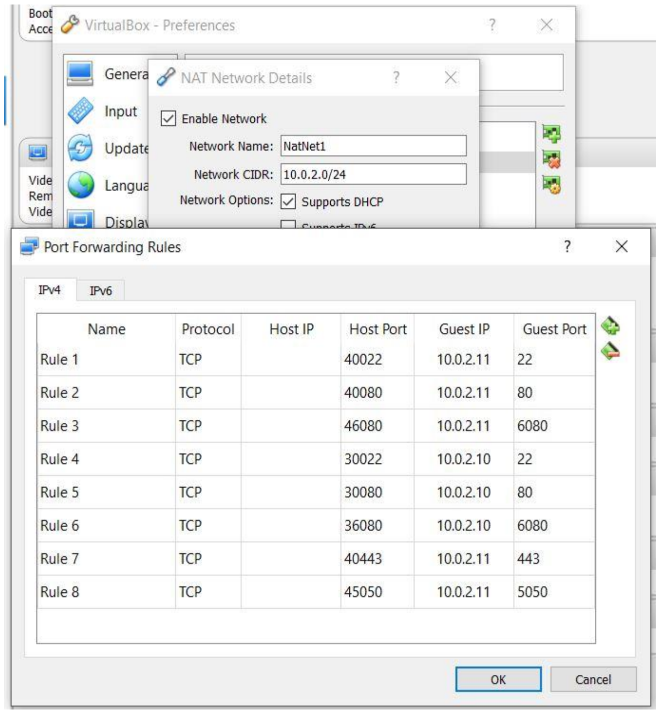


Fig5 Network Topology

By default, OpenStack networking tasks are handled by Neutron, which is a SDN networking project for OpenStack delivering networking-as-a-service. Port forwarding is done using Oracle VirtualBox to access the DevStack GUI Horizon. Port forwarding configurations in Virtual Box is as shown in below figure.



Enable Network: ✓
Network Name: NatNet1
Network CIDR: 10.0.2.0/24
Network Options: ✓ Supports DHCP

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Rule 1	TCP		40022	10.0.2.11	22
Rule 2	TCP		40080	10.0.2.11	80
Rule 3	TCP		46080	10.0.2.11	6080
Rule 4	TCP		30022	10.0.2.10	22
Rule 5	TCP		30080	10.0.2.10	80
Rule 6	TCP		36080	10.0.2.10	6080
Rule 7	TCP		40443	10.0.2.11	443
Rule 8	TCP		45050	10.0.2.11	5050

Fig6 Port Forwarding

Task 2: Use OpenStack REST API's to fetch network topology information

OpenStack SDK is a client library for building applications to work with OpenStack clouds.

Nova-API is one of the client library of OpenStack SDK. Nova API is a server daemon that serves the metadata and compute APIs in separate green threads. Through this API, the service provides massively scalable, on demand, self-service access to compute resources. These API's will be used to gain the topology information. The basic network topology of our project is as shown in the figure7.

We are using this Nova API to fetch the network topology information like Fixed IP, Floating IP, Network Namespace, Instance name etc. An example screenshot of the Active VMs present in the specified network is as shown in the figure 8.

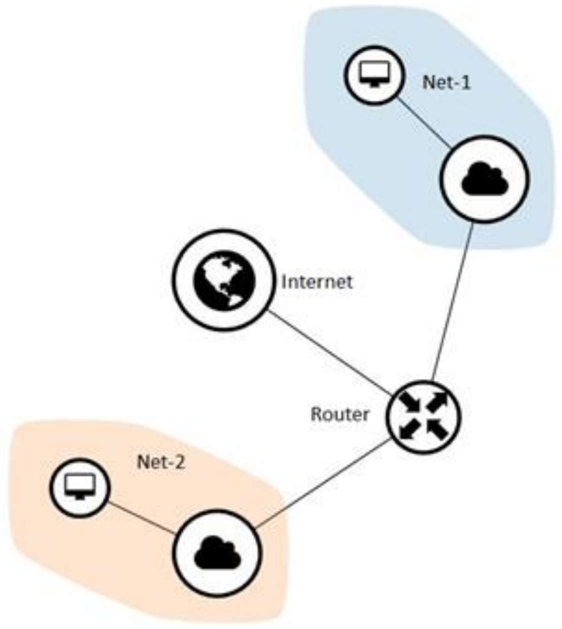
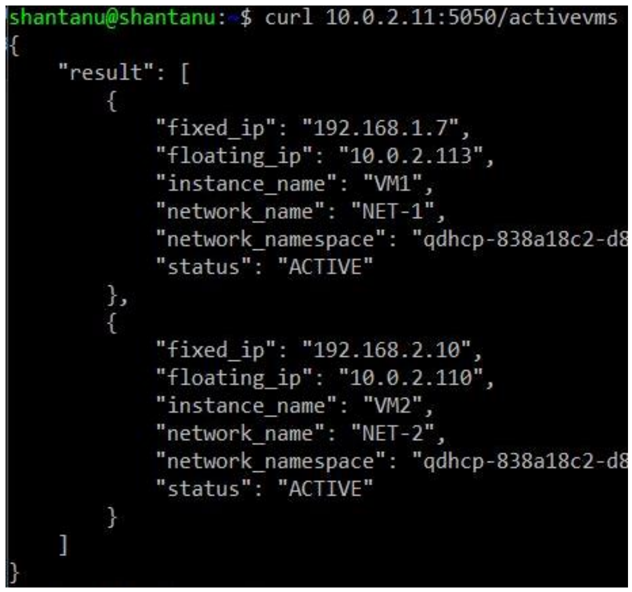


Fig7 Network Topology



```
shantanu@shantanu:~$ curl 10.0.2.11:5050/activevms
{
    "result": [
        {
            "fixed_ip": "192.168.1.7",
            "floating_ip": "10.0.2.113",
            "instance_name": "VM1",
            "network_name": "NET-1",
            "network_namespace": "qdhcp-838a18c2-d8
            "status": "ACTIVE"
        },
        {
            "fixed_ip": "192.168.2.10",
            "floating_ip": "10.0.2.110",
            "instance_name": "VM2",
            "network_name": "NET-2",
            "network_namespace": "qdhcp-838a18c2-d8
            "status": "ACTIVE"
        }
    ]
}
```

Fig8 Active VMs

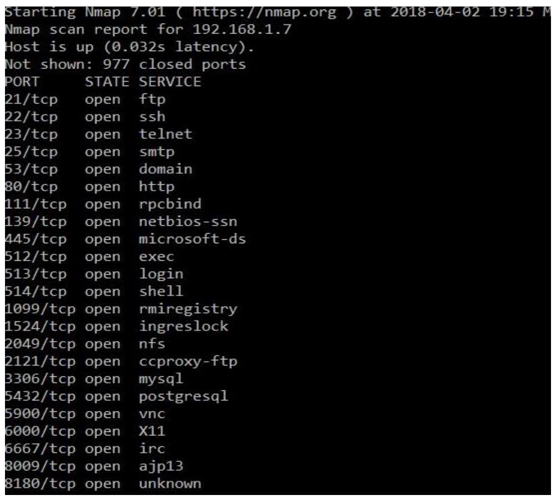
Task 3:Automating Scanning Process - Scan network for services running on virtual machines

We have written a script in python to automate the process of scanning OpenStack environments for ports which are open and are vulnerable to known attacks.

This is a two-step process and the first step of scanning the network for open ports is done as follows:

Nmap is a free and open source utility for network exploration and security auditing. The software provides several features for probing computer networks, including host discovery, service and operating-system detection.

Running Nmap script in python, we can periodically scan the specified IP ranges for open ports. This information is stored in MongoDB (as explained later), for future retrievals and comparisons.



```
Starting Nmap 7.01 ( https://nmap.org ) at 2018-04-02 19:15 M
Nmap scan report for 192.168.1.7
Host is up (0.032s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown
```

Fig9 Nmap Scan

The below lists contain some features of Nmap which can be used for further enhancement of our data begin obtained about the network being scanned.

Nmap Features:

- Host discovery – Identifying hosts on a network. For example, listing the hosts that respond TCP and or ICMP requests or have a port open.
- Port scanning – Enumerating the open ports on target hosts.
- Version detection – Interrogating network services on remote devices to determine application name and version number
- OS detection – Determining the operating system and hardware characteristics of network devices.
- Scriptable interaction with the target – using Nmap Scripting Engine (NSE) and Lua programming language.

Task 4: Automating Scanning Process – Scanning the network for services which are vulnerable using OpenVAS

OpenVAS (Open Vulnerability Assessment Scanner) is a tool designed for testing and discovery of known security issues.

To run a scan on a IP address or a range of them, we must run a cluster of five to six commands, which are not easy to remember. To make the scanning easier and to ensure changing a parameter needed for scan doesn't cause some human intervention to run a set of commands again, we wrote a python script to automate this process. This ensures that all the above-mentioned problems are mitigated and its easy for the developer to run or make changed in the future.

The following paragraphs describe the architecture and features of OpenVAS which is being used in this project.

Architecture Overview

The Open Vulnerability Assessment System (OpenVAS) is a framework of several services and tools. The core of this SSL-secured service-oriented architecture is the OpenVAS Scanner. The scanner very efficiently executes the actual Network Vulnerability Tests (NVTs) which are served via the OpenVAS NVT Feed or via a commercial feed service.

OpenVAS CLI contains the command line tool "omp" which allows to create batch processes to drive OpenVAS Manager

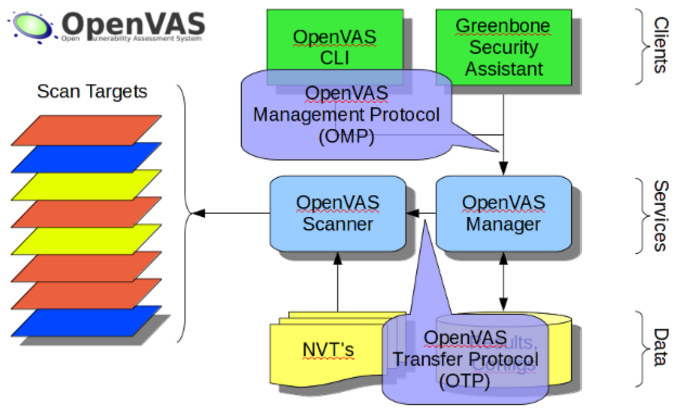
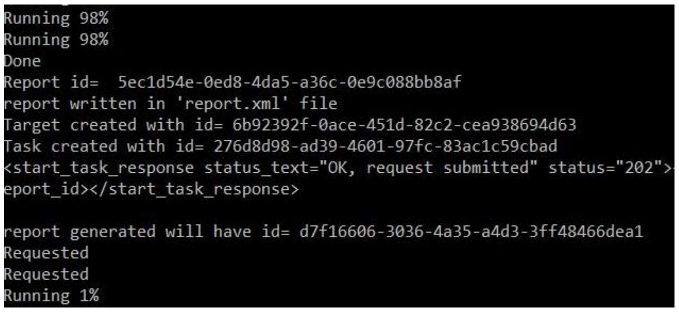

OpenVAS
Open Vulnerability Assessment System
OpenVAS CLI
Greenbone Security Assistant
Clients
Scan Targets
OpenVAS Management Protocol (OMP)
OpenVAS Scanner
OpenVAS Manager
Services
NVT's
OpenVAS Transfer Protocol (OTP)
Results, Configs
Data

Fig10 OpenVAS

Feature overview

- OpenVAS Scanner
  - Many target hosts are scanned concurrently
  - OpenVAS Transfer Protocol (OTP)
  - SSL support for OTP (always)
- OpenVAS Manager
  - OpenVAS Management Protocol (OMP)
  - SQL Database (sqlite) for configurations and scan results
  - SSL support for OMP (always)
  - Many concurrent scans tasks (many OpenVAS Scanners)
  - Flexible escalators upon status of a scan task
  - Stop, Pause and Resume of scan tasks
  - Master-Slave Mode to control many instances from a central one

    - Reports Format Plugin Framework with various plugins for: XML, HTML, LateX, etc
    - Feed status view
    - Feed synchronization
- Greenbone Security Assistant (GSA)
    - Client for OMP and OAP
    - HTTP and HTTPS
    - Web server on its own (microhttpd), thus no extra web server required
- OpenVAS CLI
    - Client for OMP
    - Runs on Windows, Linux, etc.



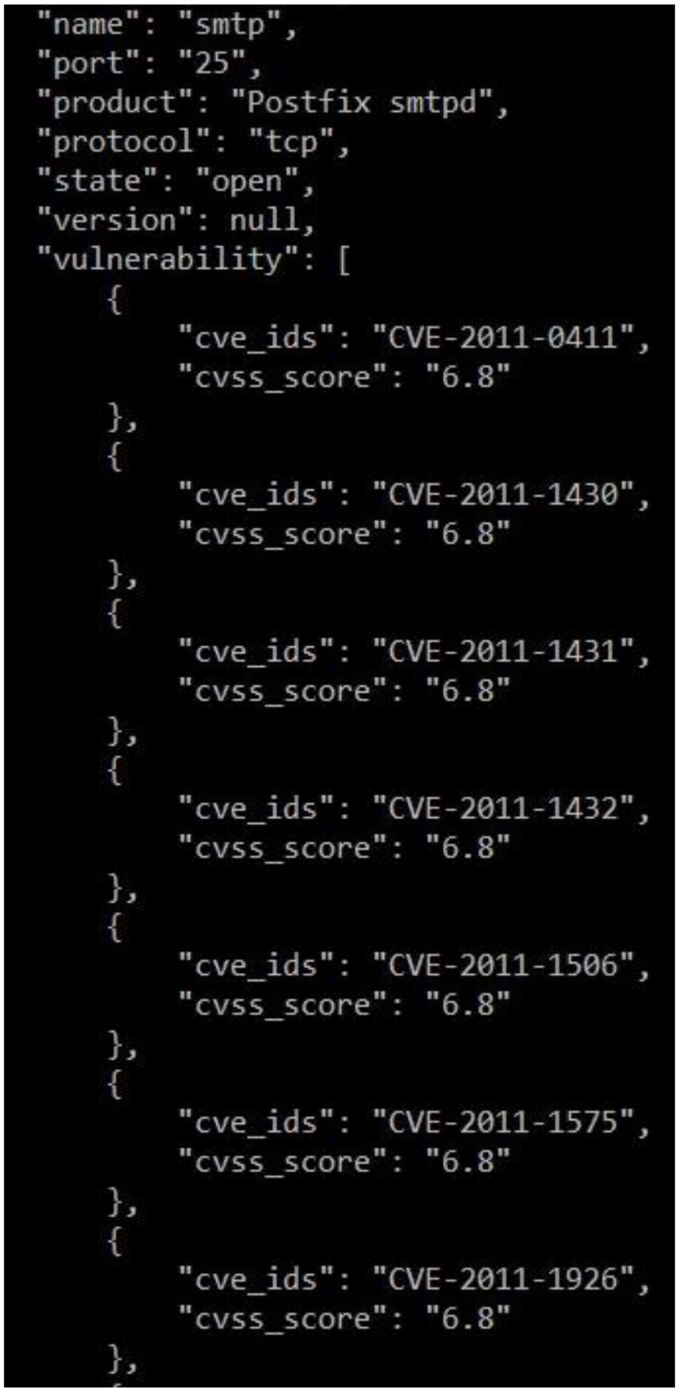
```
Running 98%
Running 98%
Done
Report id=  5ec1d54e-0ed8-4da5-a36c-0e9c088bb8af
report written in 'report.xml' file
Target created with id= 6b92392f-0ace-451d-82c2-cea938694d63
Task created with id= 276d8d98-ad39-4601-97fc-83ac1c59cbad
<start_task_response status_text="OK, request submitted" status="202">
eport_id></start_task_response>

report generated will have id= d7f16606-3036-4a35-a4d3-3ff48466dea1
Requested
Requested
Running 1%
```

Fig11 OpenVAS Scan

The above image is a screenshot of the scan progress of OpenVAS. It writes the result to a file named report.xml, which will later be converted into JSON format for easy storage, in MongoDB, and retrieval.

The below image describes the vulnerabilities present on port 25 on a system which we scanned using OpenVAS, along with their CVE ids and their corresponding CVSS score. This will be updated in MongoDB for later retrieval, when queried upon by the end user (in our case Americano).



```
"name": "smtp",
"port": "25",
"product": "Postfix smtpd",
"protocol": "tcp",
"state": "open",
"version": null,
"vulnerability": [
    {
        "cve_ids": "CVE-2011-0411",
        "cvss_score": "6.8"
    },
    {
        "cve_ids": "CVE-2011-1430",
        "cvss_score": "6.8"
    },
    {
        "cve_ids": "CVE-2011-1431",
        "cvss_score": "6.8"
    },
    {
        "cve_ids": "CVE-2011-1432",
        "cvss_score": "6.8"
    },
    {
        "cve_ids": "CVE-2011-1506",
        "cvss_score": "6.8"
    },
    {
        "cve_ids": "CVE-2011-1575",
        "cvss_score": "6.8"
    },
    {
        "cve_ids": "CVE-2011-1926",
        "cvss_score": "6.8"
    },
```

Fig12 OpenVAS result showing CVE score

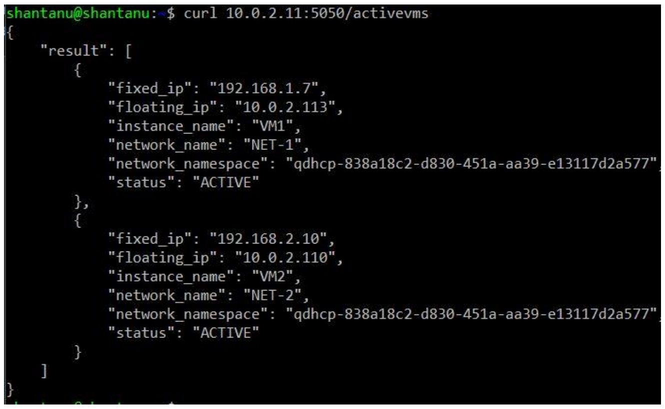
Task 5: FLASK server – Providing vulnerability information, Network information and diagnostics through REST APIs

Flask (web framework)

Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. It is BSD licensed.

Flask is called a micro framework because it does not require tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.

Using Flask, we are exposing REST APIs to end users (Americano), to query for running services, diagnostic information, and vulnerability report from our earlier scans. The detailed information about usage of APIs is given in the README file provided with the project code. The following images shows a curl request to our Flask server running on 10.0.2.11:5050 and the output is as follows



```
shantanu@shantanu:~$ curl 10.0.2.11:5050/activevms
{
    "result": [
        {
            "fixed_ip": "192.168.1.7",
            "floating_ip": "10.0.2.113",
            "instance_name": "VM1",
            "network_name": "NET-1",
            "network_namespace": "qdhcp-838a18c2-d830-451a-aa39-e13117d2a577",
            "status": "ACTIVE"
        },
        {
            "fixed_ip": "192.168.2.10",
            "floating_ip": "10.0.2.110",
            "instance_name": "VM2",
            "network_name": "NET-2",
            "network_namespace": "qdhcp-838a18c2-d830-451a-aa39-e13117d2a577",
            "status": "ACTIVE"
        }
    ]
}
```

Fig13 Curl Output showing details of active VMs

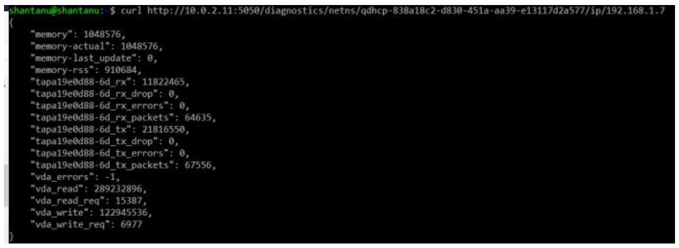
Flask was used because of being light weight and having the following features:

- Contains development server and debugger
- Integrated support for unit testing
- RESTful request dispatching
- Uses Jinja2 templating
- Support for secure cookies (client-side sessions)
- 100% WSGI 1.0 compliant
- Unicode-based
- Extensive documentation
- Google App Engine compatibility
- Extensions available to enhance features desired.

Task 6: Using Open stack APIs to show information such as CPU usage, network traffic volume on each VM

Nova-API is a server daemon that serves the metadata and compute APIs in separate green threads. Through this API, the service provides massively scalable, on demand, self-service access to compute resources. Depending on the deployment those compute resources might be Virtual Machines, Physical Machines or Containers.

We will be using Nova API for the efficient collection of metering data in terms of CPU and network costs, collecting data by monitoring notifications sent from services or by polling the infrastructure, accessing and providing the metering data through the NOVA API. The below figure shows the network diagnostic information of VM with IP 192.168.1.7.



```
shantanu@shantanu:~$ curl http://10.0.2.11:5050/diagnostics/netns/qdhcp-838a18c2-d830-451a-aa39-e13117d2a577/ip/192.168.1.7
{
    "memory": 1048576,
    "memory-actual": 1048576,
    "memory-last_update": 0,
    "memory-rss": 910684,
    "tapa19e0d88-6d_rx": 11822465,
    "tapa19e0d88-6d_rx_drop": 0,
    "tapa19e0d88-6d_rx_errors": 0,
    "tapa19e0d88-6d_rx_packets": 64635,
    "tapa19e0d88-6d_tx": 21816550,
    "tapa19e0d88-6d_tx_drop": 0,
    "tapa19e0d88-6d_tx_errors": 0,
    "tapa19e0d88-6d_tx_packets": 67556,
    "vda_errors": -1,
    "vda_read": 289232896,
    "vda_read_req": 15387,
    "vda_write": 122945536,
    "vda_write_req": 6977
}
```

Fig14 Network Diagnostic Information

Task 7: Persistent Storage of information - MongoDB

MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

Sophisticated and Efficient Access to Data
Flexibility
Data Integrity
Scalability & Performance
Enterprise Management & Integrations
Always On, Global Deployments

Fig15 MongoDB Usage

We have chosen this database because of its non-relational nature, as our scans may not produce data for key value pair, which shouldn't crash or corrupt our data. We also need versatility during development of our project, and non-relational database would be providing room for later inclusion or deletion of information being stored in our database. Moreover, the following features, makes compelling arguments for scalability of our project.

<u>Features</u>

MongoDB supports field, range queries, regular expression searches. Queries can return specific fields of documents and include user-defined JavaScript functions. Queries can also be configured to return a random sample of results of a given size.

- Indexing
- Replication
- Load balancing
- File storage
- Aggregation
- Server-side JavaScript execution
- Capped collections
- Transactions

<u>Storage:</u>

We use the following tree structure to represent data in our database. This helps us keep track of all the vulnerabilities which are present from our latest scan. We can also use this later to compare the results to check which vulnerabilities we have remediated.

```
|->network namespace
|->Scan Time
|->OS names (list)
|->oscpes (list)
|->Instance Name
|->floating ip
|->fixed ip
|->mac address
|->Network Name
|->Services (list)
  (In Each Element)
          |
          |->state
          |->product
          |->version
          |->protocol
          |->name
          |->port
          |->cpe (list or key-value pair)
          |->Vulnerability (list)
          (Each element)
                    |
                    |->cvss score
                    |->cve-id
```

Task 8: Automation of scanning, monitoring, topology change updates in Cloud Network.

The novel approach that we are taking in this project is to abstract the ad hoc scan. Our module is configured to run a cron jobs, which is automated regular periodic scans, to generate results and to store them in a database.

This script will be running in the background and whenever the end user initiates a scanning task, we will return the results from the latest scan which was completed. This ensures that the server doesn't get clogged with scan requests and not be able to perform its task.

B. *Project Task Allocation*

Member Responsible	**Task**
Shantanu Bhusari	Task1, Task2, Task3, Task7
Chandramouli J	Task1, Task2, Task4, Task8
Raghuveer Gowda H	Task1, Task2, Task5, Task6

C. *Deliverables*

It includes source code and scripts, RESTfull API's, user guide and supporting documents.

D. *Project Timeline*

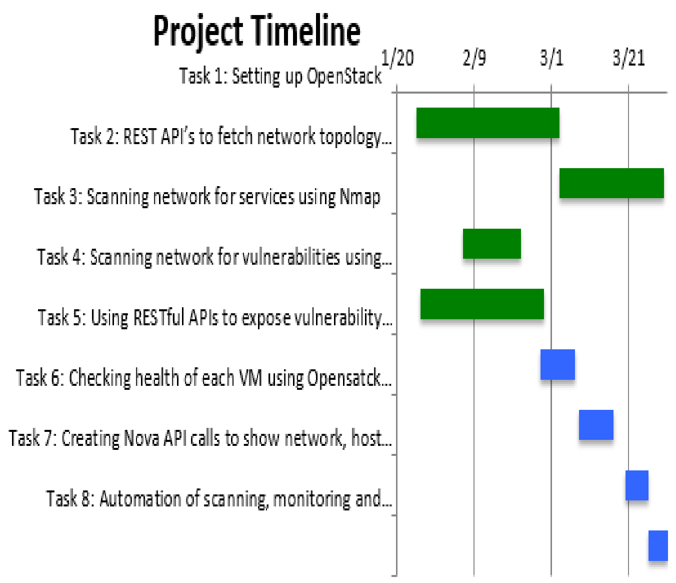


Fig16 Project Timeline Gantt Chart

IV. FUTURE SCOPE

Even though we have implemented our project with various features, we would still like to improve some parts of it. The following describes some future improvements which could be made:

1. Implement better telemetry using ceilometer
2. Use of opensource controller, such as Open Daylight controller, for fetching network information.
3. Implement scanner to scan for UDP ports as well.

V. CONCLUSION

We have successfully developed an automated network scanner module as a service to collect the network topology information which will help in identifying and mitigating the vulnerabilities present in the network. We have successfully enhanced the accessibility of our module wherein we have integrated this module with Americano module which will result in an Intelligent Software Defined Secure System.

ACKNOWLEDGMENT

We sincerely thank Dr. Dijiang Huang for giving us this opportunity, and Ankur Chowdary for mentoring and assisting us whenever required.

REFERENCES

1. https://en.wikipedia.org/wiki/MongoDB
2. https://en.wikipedia.org/wiki/OpenVAS
3. https://en.wikipedia.org/wiki/Flask(web_framework)
4. https://wiki.OpenStack.org/wiki/DevStack
5. https://docs.OpenStack.org/python-OpenStacksdk/latest/
6. https://docs.OpenStack.org/python-novaclient/pike/
7. https://flask-restful.readthedocs.io/en/latest/
8. https://pythonadventures.wordpress.com/2014/12/29/xml-to-dict-xml-to-json/
9. https://netaddr.readthedocs.io/en/latest/tutorial_01.html#support-for-non-standard-address-ranges
10. https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/