MALWARE DETECTION IN THE CONTEXT OF CLOUD COMPUTING

SAN JOSÉ STATE UNIVERSITY

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Computer Engineering
Master of Science in Software Engineering

By

Vinoth Baalaji A S, Saranya Mohan, Srinivasulu Ponduri, Pooja Takavale

August 2016

APPROVED

[Advisor's Name], Project Advisor

[Program Director's Name], Director, MS Computer Engineering

[Program Director's Name], Director, MS Software Engineering

[Department Chair's Name], Department Chair

Abstract

Malware Detection in Cloud Computing

By

[Vinoth Baalaji A S, Saranya Mohan, Srinivasulu Ponduri, Pooja Takavale]

Cloud services have increased significantly in recent times, as it reduces enterprise infrastructure cost and provides scalable solution to its customers. Cloud services rely on virtualization, to effectively share the physical resources and quickly create Virtual Machines based on the demand. A virtualization layer between the actual hardware and the virtual machines is required in order to achieve the same. With more number of services moving to cloud, the concerns over the security attacks and breaches have also increased.

With the introduction of virtualization layer in cloud system, it is challenging to detect the malware at both application and Operating System (OS) level. In the Signature based detection approach that is currently used, the signatures are collected in a repository. When a malware is identified, the database is updated with the malware signature. A drawback of this approach is that the new threats cannot be identified from the existing signature repository. Also, malware signature repository increases significantly which makes it difficult to maintain the repository over the period.

In this project, we propose a heuristic approach to detect malicious activity in cloud using machine learning approach to classify malwares. Every datacenter will consist of an agent that monitors the Process executable files (.exe) in virtual machines. The behavioral characteristics of the malwares are collected by executing them in a sandboxed environment. This behavioral data will form the input dataset for malware analysis. The machine learning algorithm uses this data to classify malicious and benign process. For e.g. to detect Trojan malware activity, the agent will collect the behavior data of the malware. If the data matches a malicious pattern, algorithm identifies it as a malware process.

Acknowledgments

The authors are highly indebted to our project advisor **Prof. Hungwen Li** for his valuable comments, suggestions and assistance in the preparation of this study. This project would not have been possible without sincere efforts by the project team. A special thanks and appreciations to the team in developing the project.

Furthermore, we would like to express our gratitude to **Prof. Dan Harkey** for the valuable comments and helping us to drive the project in right direction.

Table of Contents

List of Figures

Chapter 1 Project Overview

Introduction

In the Internet era, there is increasing popularity for cloud computing. With this there is a tremendous increase in security threats affecting the systems on Cloud. Many antivirus and security companies are developing different malware detection approaches to detect the rampantly varying malwares. Signature based approach is one of the basic approaches available but has some major drawbacks. One of main drawback is maintaining the signature database as more and more metamorphic malwares are generated. Another popular approach mentioned is behavioral based approach that detects malwares based on the behavior of the process but the issue arises when a new malware that has totally new features. It becomes difficult to detect new malwares using traditional signature and behavioral based approach.

Malware, a short form for malicious software is anything that interrupts the operation of computer systems. It can be unwanted advertisements, giving access to remote computer system without user consent or gathering personal user information like credit card details or account details without authority. If a computer system is infected with malwares it can cause serious effects to the privacy of the system and results in a security threat for computers. The type of malicious programs includes viruses, adware, hijacking software, spyware, fake security software etc. Malwares can take the form of executable code, script or active contents and each one of it affects the system in different way. In the case of spywares it is mostly in the form of embedded programs that gets installed along with other software with or without our permission into the personal computers and tends to gather information about users. Trojan looks like genuine applications which if run in the system can affect by creating a backdoor entry to the system which gives malicious programs access to confidential and personal information. Ad wares are generically advertising banners displayed in software applications while any program is running. It can get downloaded to the system automatically while browsing or through pop-up windows. Worms are very similar to virus, which has the capability of replicating functional copies of itself which can cause the same kind of damage.

We want to develop a VMM based detection approach that provides an outside-in perspective of Virtual machine functioning. We have the freedom of using various kinds of data in this approach like memory introspection, file systems, network activities and also baseline disk images. In order to maintain security in cloud infrastructure, the entire cloud system should be monitored at a single point rather than separate detection systems. The collaborated system forms the agent machine that monitors all the machines in the network. The agent monitors the system and sends alert if a threat is detected in a virtual machine. The detection mechanism that is mostly preferred for machine learning algorithms is characteristic based approach.

Various machine-learning approaches can be used for malware detection. Some of the methodologies like clustering, regression can be used for the detecting malwares. But classification algorithms like Random Forest, Naïve Bayes, SVM are more appropriate for detecting malwares using behavioral analysis. The first step will be to extract and generate a feature set from the API call logs data generated from the VMs with which we can develop a classification model for malware detection. The key feature for our dataset is identifying API call logs/DLLs that are invoked during execution in real time. We have collected samples of malwares from different malware families like Winwebsec, Zbot, Zeroaccess. In order to test our prediction results we have collected samples of benign process to generate our training dataset.

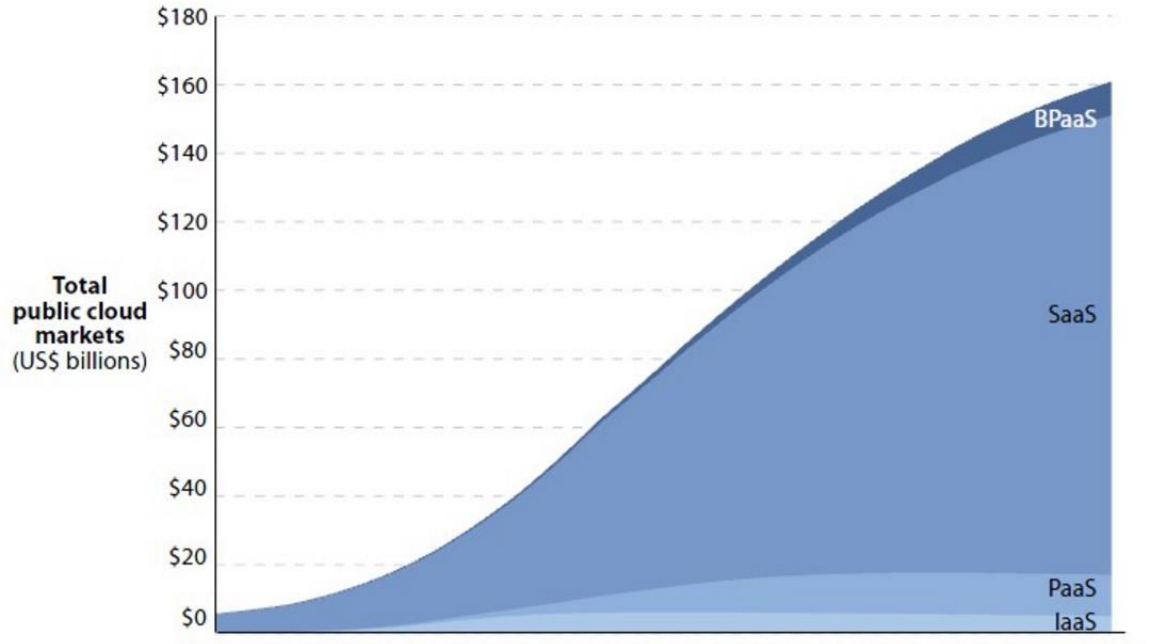
Proposed Areas of Study and Academic Contribution

This report will give more insight on various malware detection techniques and the approach that we follow is more of a data driven implementation. The entire detection systems accuracy depends on the amount of data we can collect and thus usage of some of the Big Data technologies is necessary. We are also using various virtualization technologies backed up with machine learning methodologies that can help in development of a more effective system, which can run in any cloud network. A VMM based approach requires the development of an

agent machine, which can monitor and detect multiple VMs at a time. We are trying to develop a classification model so various machine learning algorithms like SVM, KNN, Decision Tree are considered for our analysis and development.

Why Heuristics Malware detection is important

Analysts in the field of technology say that "*cloud computing is here for a long haul*". The driving growth in the field of cloud computing is due to number of benefits to all sector of companies, i.e. ability to use software from any device. These driving growth in cloud industry is a cost saving, because of outsourcing software and hardware necessaries. According to NASDAQ, an American stock exchange, investments in key strategic areas like Big Data Analytics, Security and Cloud technology is expected to increase more than 40 million dollars by 2018[12]. With this exponential increase in growth of cloud computing there will be huge market to invest in, for a system which could design the cloud environment to much more stable.



	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
BPaaS ($)	0.15	0.23	0.35	0.53	0.80	1.26	1.95	2.93	4.28	6.00	7.66	9.08	10.02
SaaS ($)	5.56	8.09	13.40	21.21	33.09	47.22	63.19	78.43	92.75	105.49	116.39	125.52	132.57
PaaS ($)	0.05	0.12	0.31	0.82	2.08	4.38	7.39	9.80	11.26	11.94	12.15	12.10	11.91
IaaS ($)	0.06	0.24	1.02	2.94	4.99	5.75	5.89	5.82	5.65	5.45	5.23	5.01	4.78

Figure 1 FORECAST: Global Public Cloud Market Size, 2011 to 2020

Source: Forrester Research, Inc.

Despite increase in growth of cloud, organizations are right to hesitate about moving to cloud without guarantee about protection. A system to detect malware in cloud is very much essential. As malware in cloud spread really fast, a single compromised document might lead to wreak havoc, when it is shared across the organization [13]. A cloud malware can lead to breach in data, a sensitive or protected data can be spread out. A cloud malware could lead to even more threats.

To remain resilient, cloud technology needs to detect not only known threats but also emerging threats that target cloud infrastructure. In an enterprise, it uses hundreds of cloud services. There is a serious security threat as enterprises send huge amount of data to high-risk services. In a survey [14], after analyzing data from 10.5 million users of cloud from different sectors like

financial, educational, tech, media and retail, the survey concluded that on an average a company sends 80GB of data to high-risk cloud services every quarter. Sky-high Networks says that, there are overall of 3,816 unique cloud services in use worldwide. Majority of these 3,816 services lack basic security features which lead to security risk. Threats in cloud are increasing with adoption, malware attacks are increasing year over year.

Threat Diversity	Cloud Hosting Provider(CHP) Vs. On-premises			
	% of customers impacted		Frequency	
	CHP	On-Premises	CHP	On-premises
App Attack	4%	16%	2.2	2.1
Brute Force	44%	49%	26.1	55.7
Malware/Botnet	**11%**	**56%**	**9.7**	**32.7**
Recon	6%	18%	2.5	6.4
Vulnerability Scan	44%	40%	13.4	9.7
Web App Attack	44%	31%	31.9	46.7

Figure 2 Statistics on Malware attacks

With these situation in current technology, where there is growth in cloud computing and same time hesitation about using it due to concerns about protection, we strongly believe the system that is being developed will attract a lot of investors towards our project.

“To become a viable choice for enterprises, cloud computing must widely adopt some security practices and standards”.

Current State of the Art

Cloud services have increased significantly in recent times, as it reduces enterprise infrastructure cost and provides scalable solution to its customers as a result of which most enterprise applications are moving to cloud platform. With the increase in popularity comes the

increasing chance of security threats. Virtualization is the key technology, which helped in increasing the number of clients under a physical server and in turn helps in developing infrastructure as a Service (Iaas). VMs create an ideal environment for malware propagation so in order to provide protection in cloud requires a good coordination between the virtualized environment for threats to be reliably detected and handled.

In cloud computing there are a lot of security issues which can be concerning, of which insecure interface and APIs, share technologies issue are directly related to malwares. A malware running on one VM can directly execute code or access data of another VMs through interfaces and APIs. New forms of worms can spread by writing through memory due to shared physical memory of VMs. Some of the conventional malware detection techniques includes signature based detection and behavioral based detection. Both signature based detection and behavioral detection works by analyzing the malware instructions and identifying the instructions that comprises of the signature or analyzing the behavior of the malware sample through observation of running processes. All the existing methods fails to detect variants of known malware and previously unseen malware.

We try to overcome the drawbacks in the existing approaches by using machine learning algorithms. How data mining works for malware detection is by applying statistical and machine learning algorithms on a set of features derived from malicious and clean programs and using statistical methods to classify various threats. The classification algorithms developed will be able to construct a model to represent the benign and malicious classes. Both supervised and unsupervised learning can be used for our analysis. In supervised learning a labeled training set is required to build the class models. Some of the supervised learning approaches that we are planning to use are Naive Bayes, SVM, and Decision Tree etc.

Naive Bayes is one of the best learning algorithms for text categorization and it is based on the Bayes rules. The Bayes Theorem works by finding probability of an event's occurrence given the

probability of another event that has occurred already. Support Vector machines text documentation classification (SVMs) aim at searching for a hyper plane that separates the two classes of data with largest margin. KNN classifies objects based on closest training instances in the feature space and object are classified based on a majority vote of its k nearest neighbors at closest distant from the object. Unsupervised learning approaches is anomaly-based detection techniques and uses the knowledge of what is considered as good behavior to identify malicious programs. Anomaly based detection makes use of certain rule set for considering acceptable behavior of programs. Programs that are violating the set of rules declared are considered malicious.

Existing Malware Detection Techniques

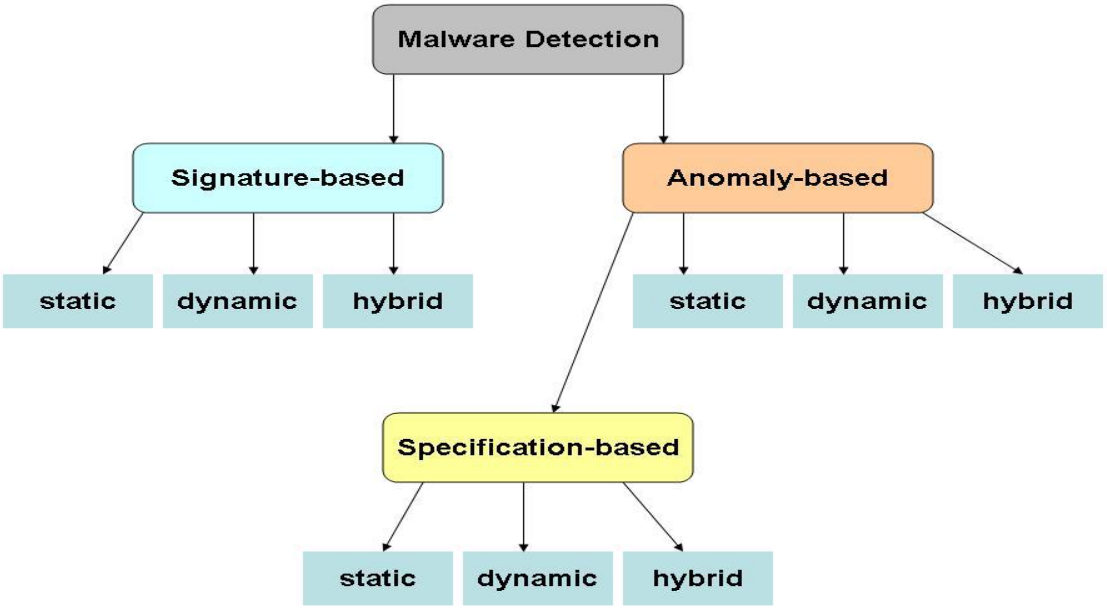


Figure 3 Existing Malware detection techniques

Signature Based

In signature-based detection, it develops model on its own which resembles the malware behavior. The signature data collection will represent the Knowledge of the detector. It acts as

the database storage that stores the signatures of the malwares. All the detection signatures is stored in the Repository.

Only, people with expertise can develop the signatures to combat malware effects in future. The signatures represent the malicious behaviors exhibited by the malwares. After, developing the signature of the behavior, it is added to the repository.

The major Drawback of the signature based detection technique is that it cannot detect the "Zero Day Attack". Zero day attacks are the attacks whose signatures are not present in the repository. Another, drawback of this attack is that it involves the human expertise in development of the signatures. It usually takes very long time to develop the signatures and human errors are also possible. When the malwares are generating themselves at a faster rate it becomes highly impossible to develop the signature in a very short period of time.

Static Signature Based

In static based technique, the program is monitored in the sequence of code to detect its maliciousness. The main goal is to represent the behavior of the program. In static analysis it gives the approximate estimation of how the code will behave in the dynamic situation.

In signature based, there is usually a sequence of code, which are responsible for the bringing about the malicious behavior. The key advantage of static based is that the maliciousness can be accurately determined without having it to execute.

Dynamic Signature Based

In dynamic analysis, the information can be gathered solely when the program is executing. And upon the execution the maliciousness can be determined. It look for the patterns to determine the maliciousness of the program.

Hybrid Signature Based

Static and dynamic properties are used together to determine the maliciousness of the program.

Anomaly Based

Anomaly based detection techniques decides the maliciousness of the program upon the inspection. It uses its knowledge to decide the malicious behavior of the program. Anomaly based detection mostly occurs in two phases – A training (learning) phase and the Detection (Monitoring) phase. It will train itself about the behavior of the malwares in the training phase.

The Key advantage of the Anomaly based detection is that it can detect the Zero-day attacks and has the ability to detect the new malwares.

There are two fundamental disadvantage of this technique:

1. It has high false alarm rate.
2. It is very complex to know what features should be earned in the training phase.

Static Anomaly Based

In Static based anomaly detection system, the file structure characteristics of the program are considered and are under inspection to detect the malicious code of the program.

A key and major advantage of the static technique is that its use can make it possible to detect the malware without it getting executing on the host system the malware carrying program.

Dynamic Anomaly Based

In the dynamic anomaly based technique, to detect the malicious code, information is gathered from the program while execution of the program. During the execution of the program, information is been gathered and in the detection phase it monitors the program that is under inspection.

Hybrid Anomaly Based

Static and dynamic properties are used together to determine the maliciousness of the program.

Specification based

Specification based anomaly detection technique is also a technique which is based on the anomaly detection technique. In this, it tries to address the typical high false rate which is generally associated with the anomaly based techniques. Specification based majorly focusses on approximating the requirements for the application and system instead of focusing more on implementation of it.

In specification based detection, the training phase is the attainment of the rule set, which usually specifies all the valid behavior for any program that it will exhibit for the system that is protected or the program is under inspection.

The only main disadvantage is that it is really very complex and difficult to maintain the entire set of the valid behaviors of the programs and system.

Static Specification Based

In the static specification based detection system, to determine the maliciousness the structural properties of the PUI are used.

Dynamic Specification based

In this detection technique, the behavior is observed during the runtime to determine the maliciousness of the program.

Hybrid Specification based

Static and dynamic properties are used together to determine the maliciousness of the program.

Chapter 2 Project Architecture

Malware Detection in Cloud Computing - (Heuristic approach)

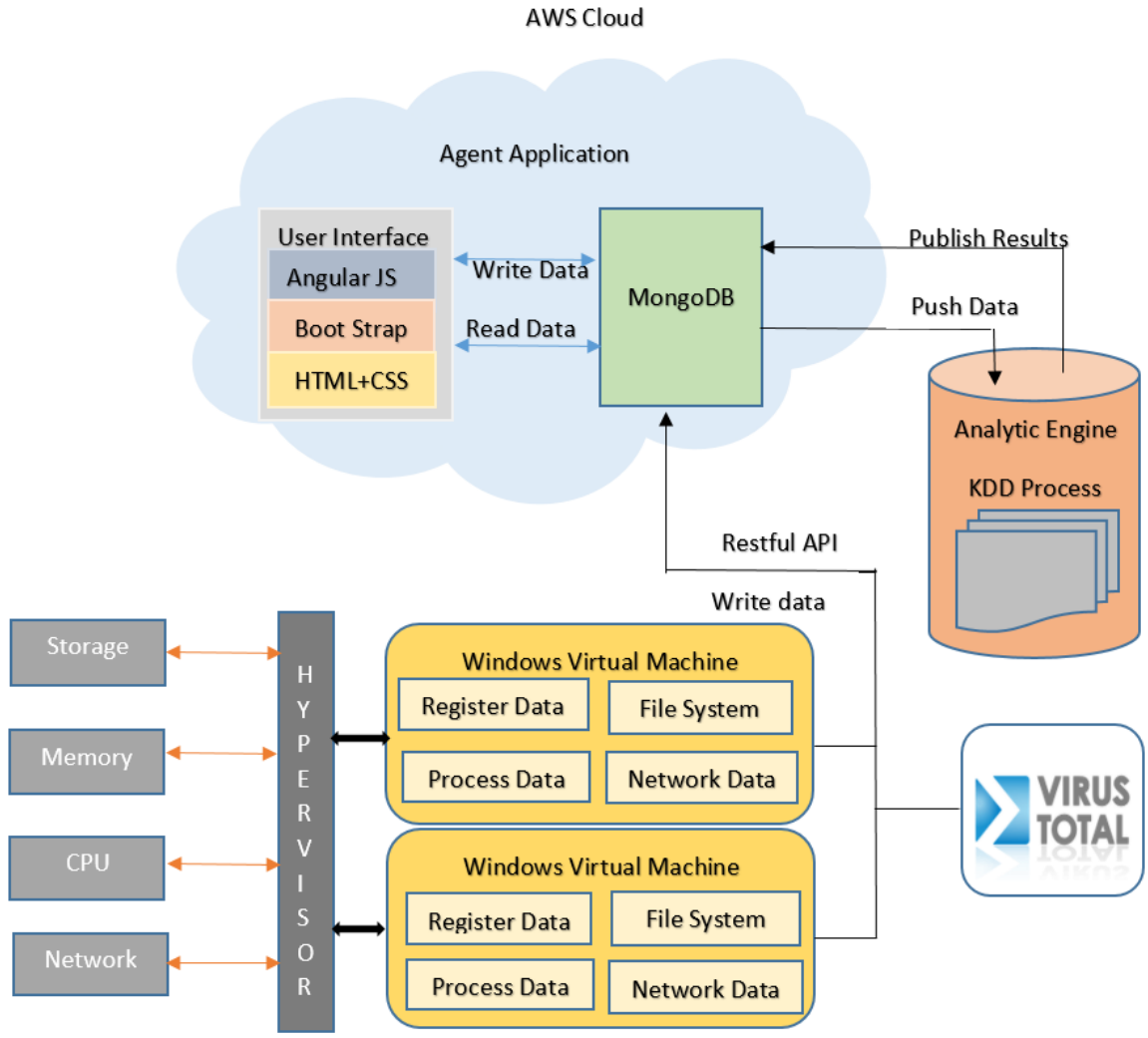


Figure 4 Project Architecture

Introduction

The proposed malware detection system is designed to detect malwares based on real time data obtained from the Virtual machines. The detection system consists of an agent application that monitors the activity of virtual machines. The Agent acts as an interface between virtualized layer and the analytics system. Since, the malwares are more prominent in

Windows machines, we will create Virtual machines with Windows operating system installed on them. The Windows's Power shell utility will be used to read the operating system related information from the Virtual machines. The project infrastructure will be setup on the workstation identified in the CMPE lab. The workstation will be virtualized using VMWare's vCenter application. The architecture consists of three core modules namely Virtualization module, Agent module and the Analytics module. The modules communicate using RESTful services to interact with each other.

Working

The detection system is expected to act based on the levels of malware activity. The primary users of our detection system will be administrators who can visualize the performance data from vCenter server, check the malware scan status, and identify malware information in virtual machines. The application will be designed such that an administrator, who has access to a vCenter, can use our system to create agents and monitor the virtual machine activity. The user logs into the application using the vCenter credentials such as username, password and vCenter name. The vCenter client validates the request and extracts the list of Virtual machines from the vCenter in a JSON format. The user first creates an agent to add the Virtual machines under it. The agent application collects Virtual machine specific information such as Guest OS name, Memory, Storage and name of the virtual machine. The component view of the architecture is mentioned below.

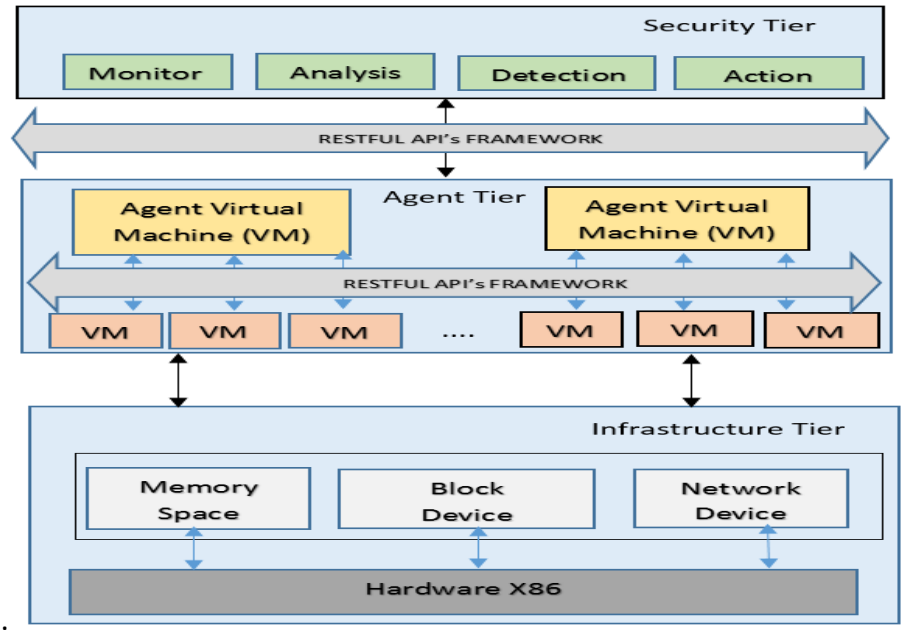


Figure 5 Component View of Architecture

With the help of performance counter utility in vCenter, the agent application collects and displays the performance of the Virtual machine in a timely manner. The performance data will be displayed in the form of pie charts and pulse charts. When the user initiates a scan button, the Agent starts collecting the data from the Operating system. The data will be stored in MongoDB and be sent to Analytics engine for malware detection. Based on the results obtained, the Agent application will alert the administrator to take necessary actions.

Malware Families

Out of the vast malware families impacting the Windows machines, the following are some of the most common ones. As part of our implementation we will be detecting the following malware families in the cloud environment based on the data obtained from VMs. Though Rootkit malware is more dangerous malware, it is not in our scope due to the fact that they are not academically proved on Type-1 hypervisor (Bare Metal hypervisor).

WinWebSec:

This is a Trojan, unlike viruses these don't self-replicate. These are spread throughout system manually where they are beneficial and wanted. These malware infect the system through security exploitation.

Characteristics:

Upon execution, Trojan connects to the following URL's

- http://123.108.[Removed].42/api/urls/?ts=62f060bdeb24ad011c3c59611c4e49d77ce8d721&affid=51800
- http://123.108.[Removed].42/api/dom/no_respond/?ts=62f060bdeb24ad011c3c59611c4e49d77ce8d721&token=fya14oiYU&affid=51800&ver=3070033&group=sca

After execution, it creates the following files in below location:

- %Allusersprofile%\Application Data\004714CB29509246000000047148896C8\004714CB29509246000000047148896C8
- %Allusersprofile%\Application Data\004714CB29509246000000047148896C8\004714CB29509246000000047148896C8.exe
- %Allusersprofile%\Application Data\004714CB29509246000000047148896C8\004714CB29509246000000047148896C8.ico
- %Userprofile%\Desktop\System Care Antivirus.lnk
- %Userprofile%\Start Menu\Programs\System Care Antivirus\System Care Antivirus.lnk

And creates directories in the below location to the system:

- %allusersprofile%\Application Data\004714CB29509246000000047148896C8
- %userprofile%\Start Menu\Programs\System Care Antivirus

Figure 6 Winwebsec Malware characteristics - I

Upon execution, the Trojan adds the following registry keys to the system:

- HKEY_USER\S-1-5-21-436374069-1757981266-839522115-1003\Software\Microsoft\Installer
- HKEY_USER\S-1-5-21-436374069-1757981266-839522115-1003\Software\Microsoft\Installer\Products
- HKEY_USER\S-1-5-21-436374069-1757981266-839522115-1003\Software\Microsoft\Installer\Products\004714CB2B509446000000047148898C8

Upon execution, the Trojan adds the following registry values to the system:

- HKEY_USER\S-1-5-[VARIES]\Software\Microsoft\Windows\CurrentVersion\RunOnce\004714CB29509246000000047148896C8 = "%allusersprofile%\Application Data\004714CB29509246000000047148896C8\004714CB29509246000000047148896C8.exe"

Figure 7 Winwebsec Malware characteristics - II

Source: McAfee Threat Center

Ramnit:

These malware comes under the category of worms that spreads through removable drives. It allows the attacker to access the compromised system remotely. These malware infects the EXE, DLL, HTM and HTML.

Characteristics:

File creation

The threat may create the following files on the compromised computer:

- %UserProfile%\Start Menu\Programs\Startup\[RANDOM CHARACTERS].exe
- %UserProfile%\[RANDOM CHARACTERS].log
- %SystemDrive%\Program Files\[RANDOM CHARACTERS]\[RANDOM CHARACTERS].exe
- %ProgramFiles%\MNetwork
- %CurrentFolder%\[INFECTED FILE NAME]Srv.exe
- %DriveLetter%\autorun.inf
- %SystemDrive%\Documents and Settings\All Users\Application Data\[EIGHT PSEUDO-RANDOM CHARACTERS].log
- %UserProfile%\Application Data\[EIGHT PSEUDO-RANDOM CHARACTERS].exe
- %UserProfile%\Local Settings\Temp\[EIGHT PSEUDO-RANDOM CHARACTERS].sys
- %UserProfile%\Local Settings\Temp\[EIGHT PSEUDO-RANDOM CHARACTERS].exe
- %SystemDrive%\Documents and Settings\All Users\Application Data\[RANDOM FILE NAME].log

Registry subkeys/entries created

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security Center
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Security Center\Svc

Registry subkeys/entries deleted

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- HKEY_LOCAL_MACHINE\SOFTWARE
- HKEY_LOCAL_MACHINE\SYSTEM
- HKEY_LOCAL_MACHINE\HARDWARE
- HKEY_CURRENT_USER\SOFTWARE

Registry subkeys/entries modified

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\wscsvc
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\"Userinit" = "%Windir%\system32\userinit.exe,,%SystemDrive%\Program Files\[RANDOM CHARACTERS]\[RANDOM CHARACTERS].exe"

Figure 8 Ramnit Malware characteristics

Source: Kaspersky security center

Zero access

This is a Trojan that hides itself using advance rootkit. It creates its own hidden file system, downloads more malware and opens back door on the compromised computer.

Characteristics:

Registry subkeys/entries created

- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\[FILE NAME OF INFECTED DRIVER]\"ImagePath" = "*"
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\[FILE NAME OF INFECTED DRIVER]\"Type" = "1"
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\[FILE NAME OF INFECTED DRIVER]\"Start" = "3"

Figure 9 Zeroaccess Malware characteristics

Source: Symantec security center

SMART HDD

This malware is a misleading application that seems to be legitimate antivirus detection software.

Characteristics:

- HKEY_CURRENT_USER|\Software\Microsoft\Windows\CurrentVersion\Run
  ch1+6T~]V&zN, = %AllUsersProfile%\Application Data\ch1+6T~]V&zN,.exe
- HKEY_CURRENT_USER\software\Microsoft\Internet Explorer\Main\
  Use FormSuggest = Yes
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings
  WarnOnZoneCrossing = 0×00000000
  WarnonBadCertRecving = 0×00000000
  CertificateRevocation = 0×00000000

Figure 10 SMARTHDD Malware characteristics

Source: Kaspersky technical Support

ZBot

This is a Trojan malware, where attacker steals information from compromised system. It also downloads the configuration files and gets updated from the internet.

Characteristics:

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\"Userinit" = "%System%\userinit.exe, %System%\sdra64.exe"
- HKEY_CURRENT_USER\ SOFTWARE \Microsoft\Windows\CurrentVersion\Run\"userinit" = "%UserProfile%\Application Data\sdra64.exe"

Figure 11 Zbot Malware characteristics

Architecture Subsystems

Module: Virtual Machine Instances

This module is the primary source to generate real time data from Virtual machines for analytics. Since, the malware detection is based on cloud computing, it is essential to have a virtualized hardware environment underneath. Virtualization application VMWare is used for the same. VMWare is installed on one of the workstation to virtualize the hardware. Hardware components include Storage, Memory and CPU. VMWare is installed in the workstation. For the implementation purpose, two or three Virtual machines will be installed on the VMM. Each VM will consist of certain storage capacity and 2GB of RAM. Windows 7 Operating System will be installed in each Virtual machine in order to simulate the malwares and collect data.

Windows Power shell utility is powerful tool to execute commands required to collect data from Operating systems. The shell commands will be created in a bat file. The .bat file will be invoked from the java program and result will be sent as a response to MongoDB instance in the Agent application.

Module: Agent System

The Agent module is the core element of the malware detection system. The Agent module has to be robust, scalable and efficient to connect to any Virtual Machine instance module on a vCenter. The Agent module controls the data flow between data layer and

analytics layer. The Agent module consists of user interface that enables the users to connect to any vCenter.

The Agent module will be capable of performing the following operations

- Create an agent per VMM
- Add/Remove Virtual machines from an agent
- Display list of VMs with VM Name, Hostname, Operating system name, Memory and Storage capacity information.
- Display options to user to select the kind of information to scan like registry, process, network and performance counters.
- Display performance counter values such as Disk usage, network usage, CPU usage in MHz for every Virtual machine added under the agent.
- Initiate scan request on the virtual machine when selected user.
- Send data to analytics layer to detect malware.

Module: Agent database

The Agent machine comprises of MongoDB instance installed in it. It consists of three documents namely Application Data, Virtual Machine Data and User Data. The Application Data document stores the operating system from each Virtual Machine. The Virtual Machine Data consists of virtual machine information along with the host name and data center information. The User Data will consist of user profile related information such as the type of information to scan under each virtual machine. Finally, each VM will consist of its own document in MongoDB to write the data. The MongoDB instance will be setup on MongoLab.

Module: Analytics engine

The analytics engine is responsible to analyze large amount of unstructured and non-relational data. The data set comprises of a benign data set and a malware data set. The preprocessor in the analytics engine cleanses the data before generating the feature vector. The feature vector is generated based on the malware characteristics that were identified.

Feature vector usually consists of binary information 0's and 1's. Different combination of binary digits will form the feature vector data. In order to differentiate malware dataset from benign dataset, we will be generating two feature vectors to detect malwares.

The data obtained in the form of feature vector forms the input to the machine-learning algorithm. Today, lots of machine learning algorithms are readily available, each with a unique purpose. All that we have to do is feed some data to algorithm and it obtains the results. In our implementation, we will be using the classifying algorithm namely SVM classifier. The reason behind choosing this algorithm is because the dataset is benign and malicious. Once the algorithm is trained to learn the benign dataset, any dataset that doesn't match the benign will be considered malicious. Having said that, it is also important to reduce false positives. The working of machine learning algorithm is mentioned in the figure below

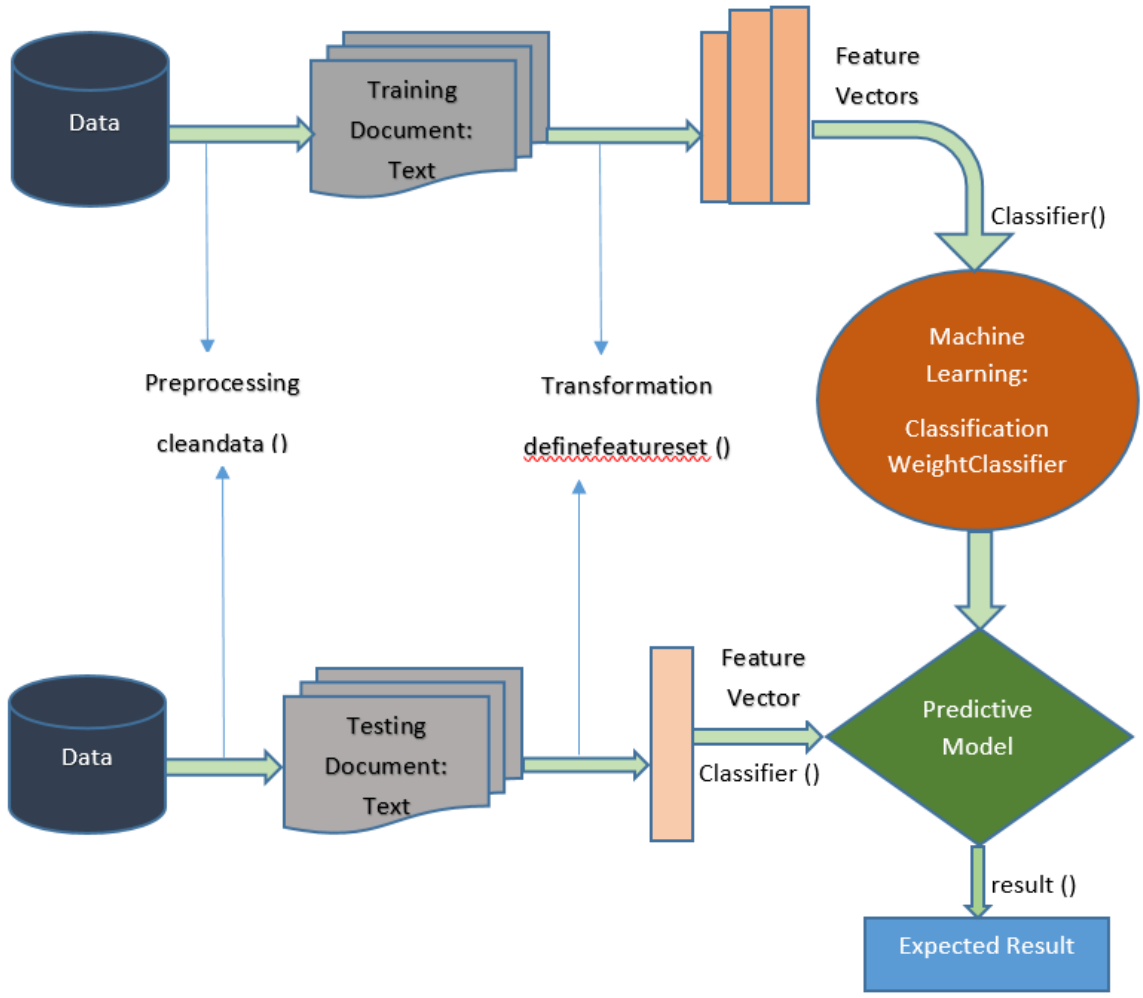


Figure 12 Machine learning flow

False positives are nothing but recognizing a benign process or data to be malicious. Let us consider the following example. The characteristics of a malware can be identified as C1, C2, C3, and C4 …. Cn. The set of characteristics C is identified to be a malware M. It can be denoted as

M consists of characteristics C_1, C_2, C_3, C_4……..C_n

There can be another benign process B with subset of characteristics namely C_2, C_3, C_4, C_{n+1}. Now, though characteristics of benign process B are a subset of malware characteristics M, B should not be recognized as a malware. Thus, the machine-learning algorithm should give

promising results by reducing false positives. In order to reduce false positives, we need to generate large data sets.

The predictive model analyzes the result set from benign and malicious dataset. The expected result will be sent to the Agent. The Agent application determines the expected result and notifies the administrator.

Chapter 3 Technology Descriptions

Client Technologies

This section contains various client side technologies used in the project. It includes Angular JS, HTML/CSS, Highcharts and RESTful API

Technologies Used	JAVA, Spring Framework, Angular JS, HTML/ CSS, Mongo DB, RESTful API, Highcharts, n node.js
Run Time Environment/Server	JRE8, JSON Libraries, Windows and Linux (ISO images), Apache Tomcat
Development Tools	Eclipse, MongoLabs, vSphere client
Cloud Hosting Environment	vCenter server on Windows server,AWS EC2
Testing Tools	JUnit, Jasmine (Behaviour driven testing)

Middle-Tier Technologies

This section contains various middle tier technologies that are used to generate data for machine learning algorithm. It includes python, python libraries to get OS data, Remote Machine WMI, VirusTotal API, VMWare SDK (SOAP WS) and Sandbox analyzer.

Technologies Used	Mongo DB, Python, Python libraries, VirusTotal API, VMWare SDK (SOAP WS), VIX API
Run Time Environment/Server	Python 2.7, JRE8, JSON Libraries, Apache

	Tomcat
Development Tools	PyCharm, MongoLab, vSphere client
Cloud Hosting Environment	vCenter server on Windows server, AWS EC2

Data-Tier Technologies

This section contains various data tier technologies used in the data-tier. It includes python libraries for Machine learning algorithms, VMWare infrastructure and JSON

Technologies Used	JAVA, Python,Sci-Kit Learn Package, Mongo DB
Development Tools	Eclipse, Pycharm, MongoLabs
Cloud Hosting Environment	vCenter server on Windows server, AWS EC2
Testing Tools	JUnit, Jasmine (Behaviour driven testing)

Chapter 4 Project Design

The following artifacts explain our project design in terms of classes, actors and sequence of program flows.

- Class diagram
- Entity Relationship diagram
- Activity diagram
- Use case diagram
- Sequence diagram

Class Diagram

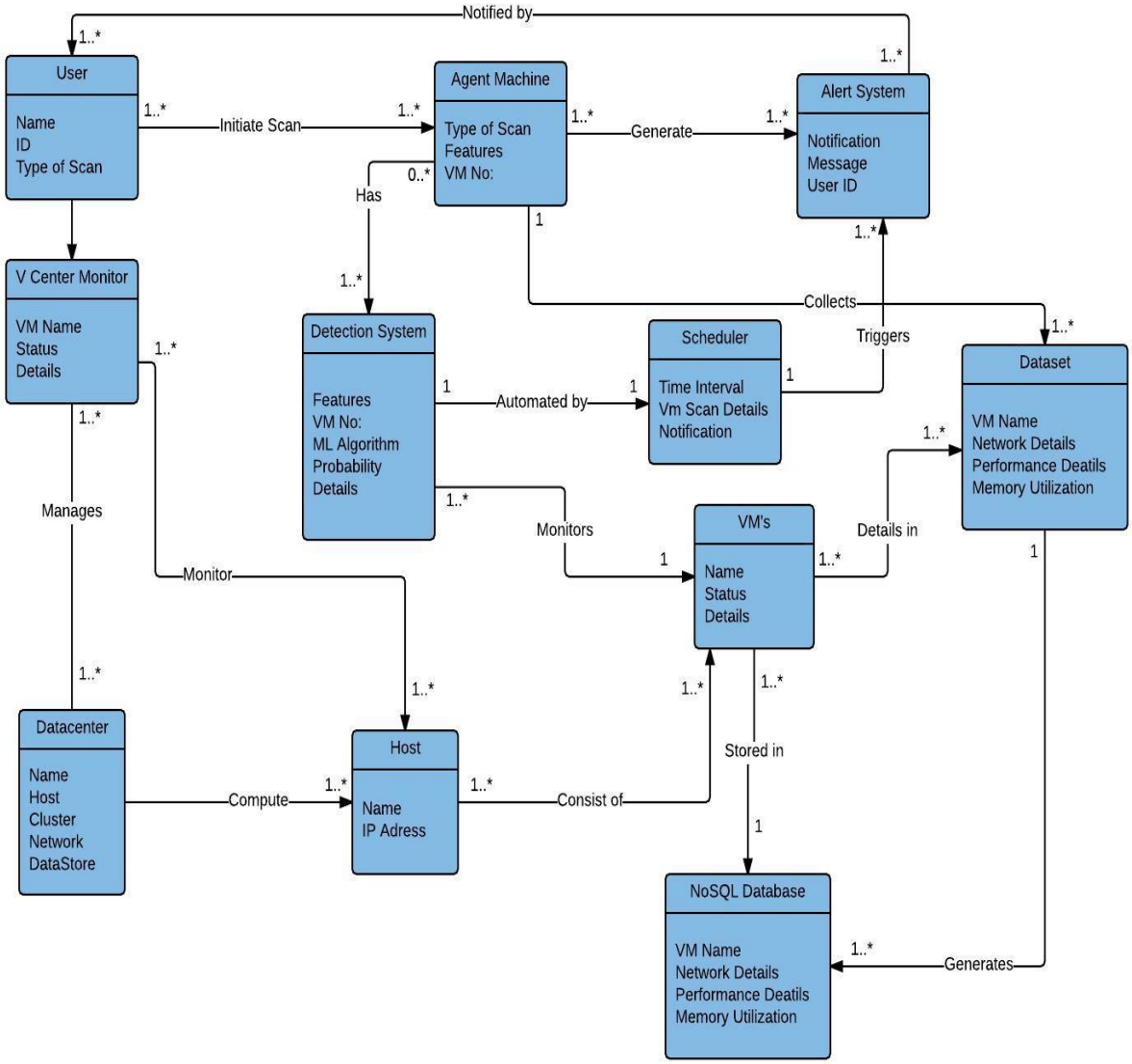


Figure 13 Class Diagram

ER Diagram

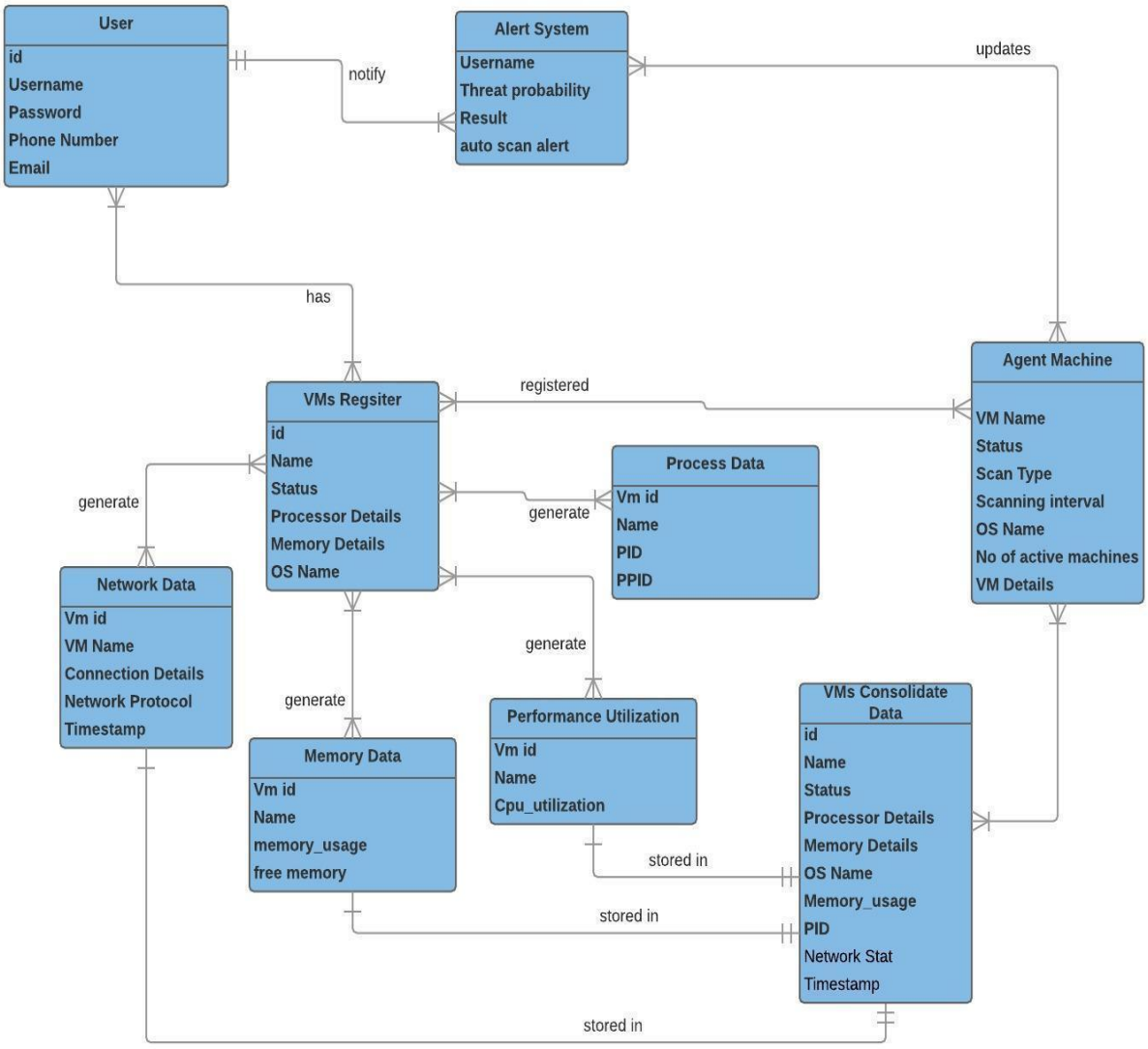
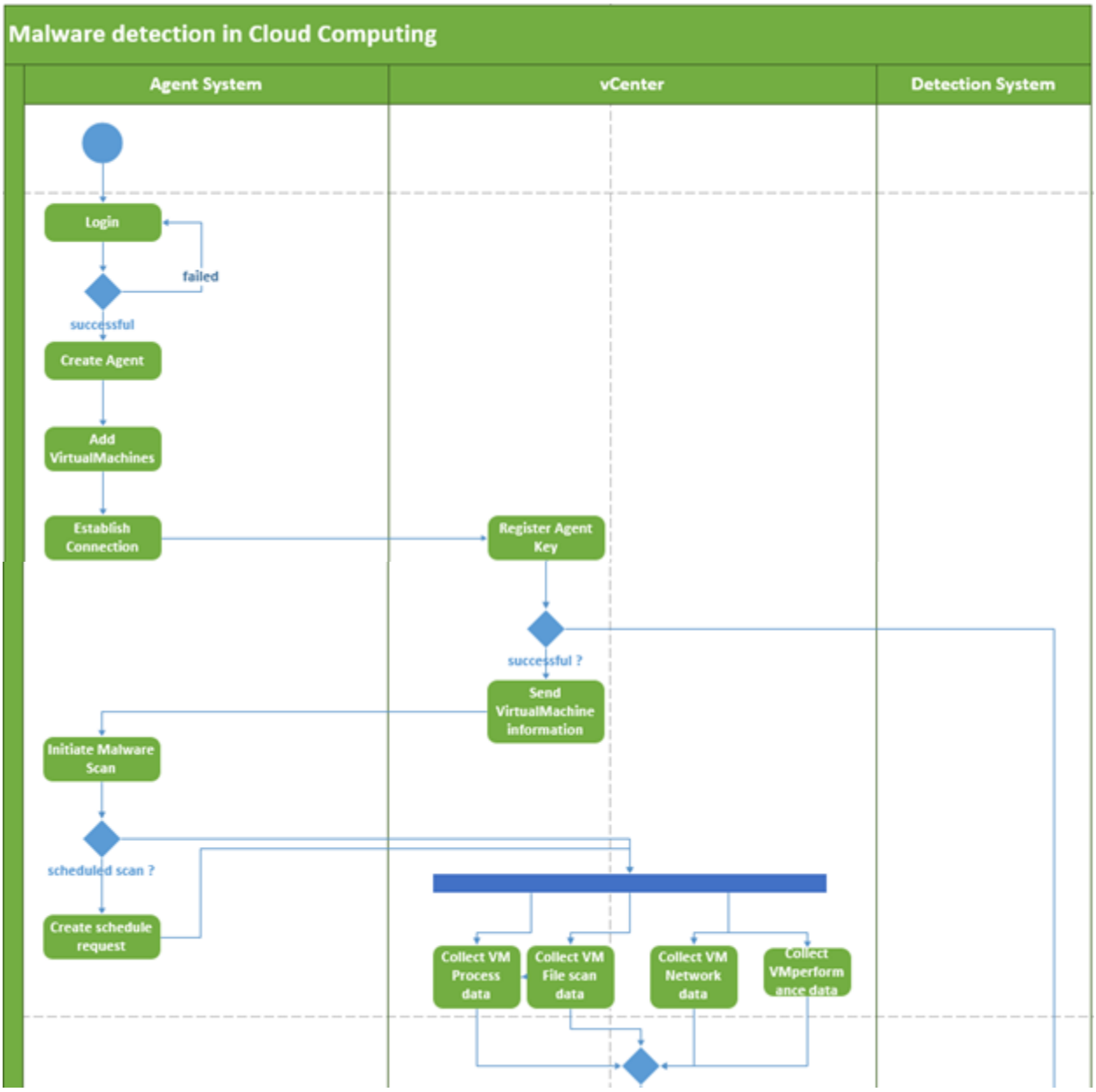


Figure 14 ER Diagram

Activity Diagram



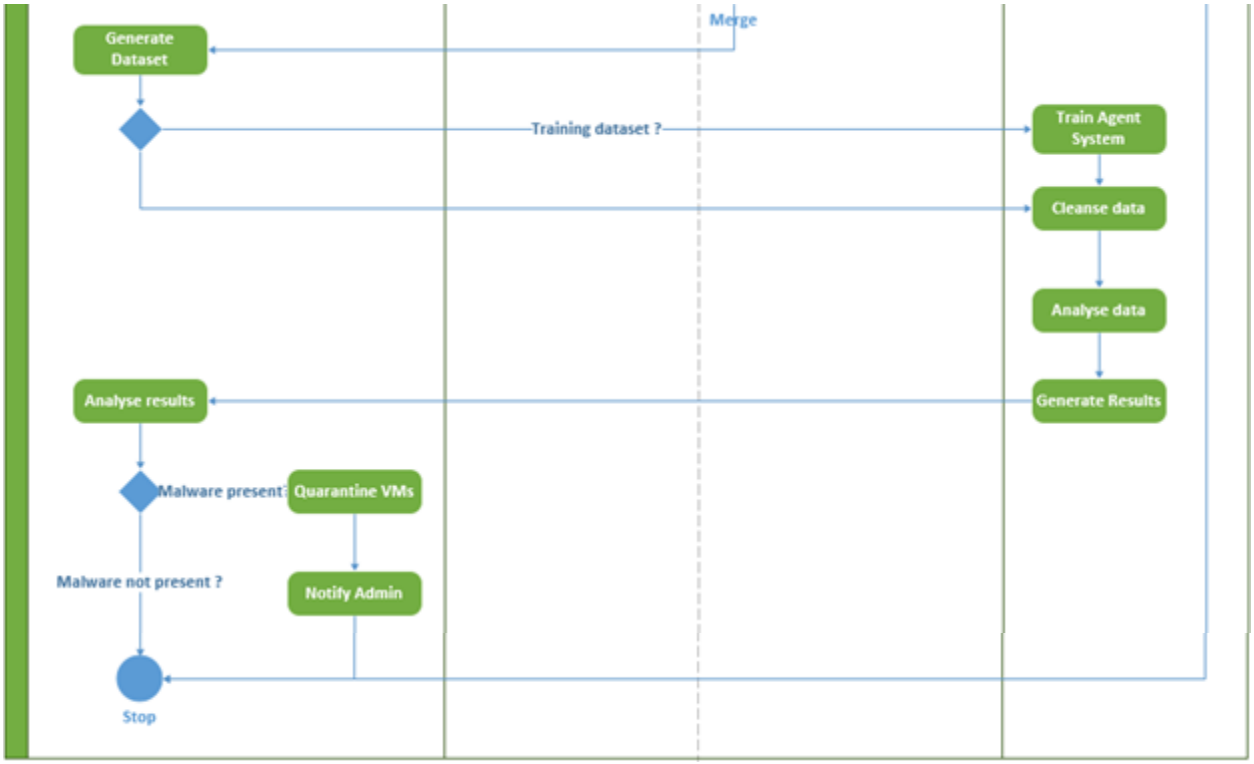


Figure 15 Activity diagram

Use Case Diagrams

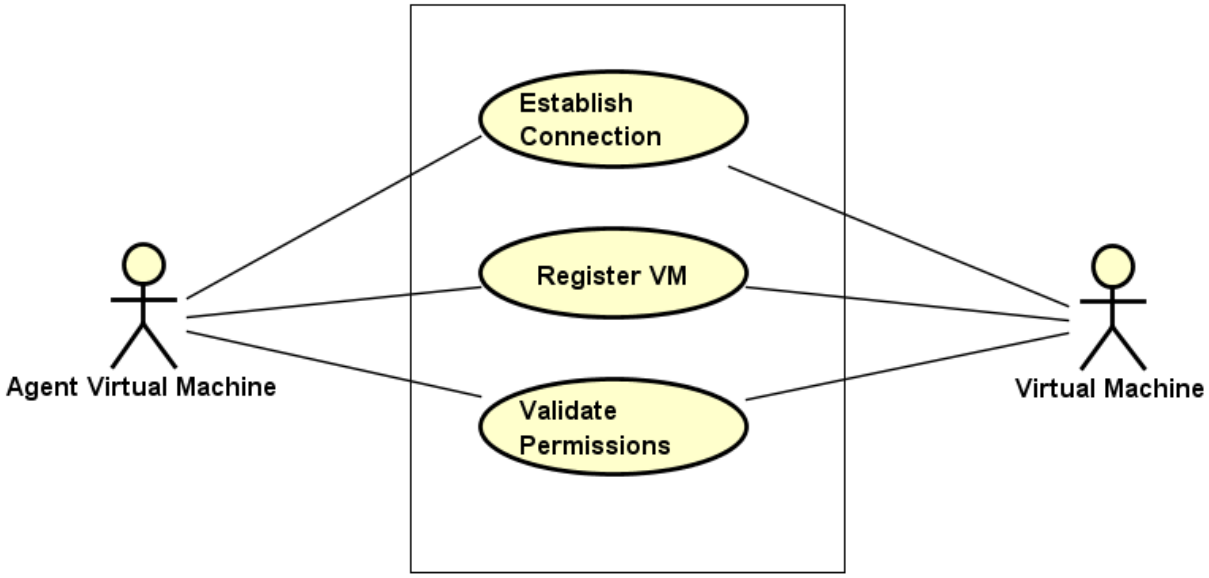


Figure 16 Use case diagram 1

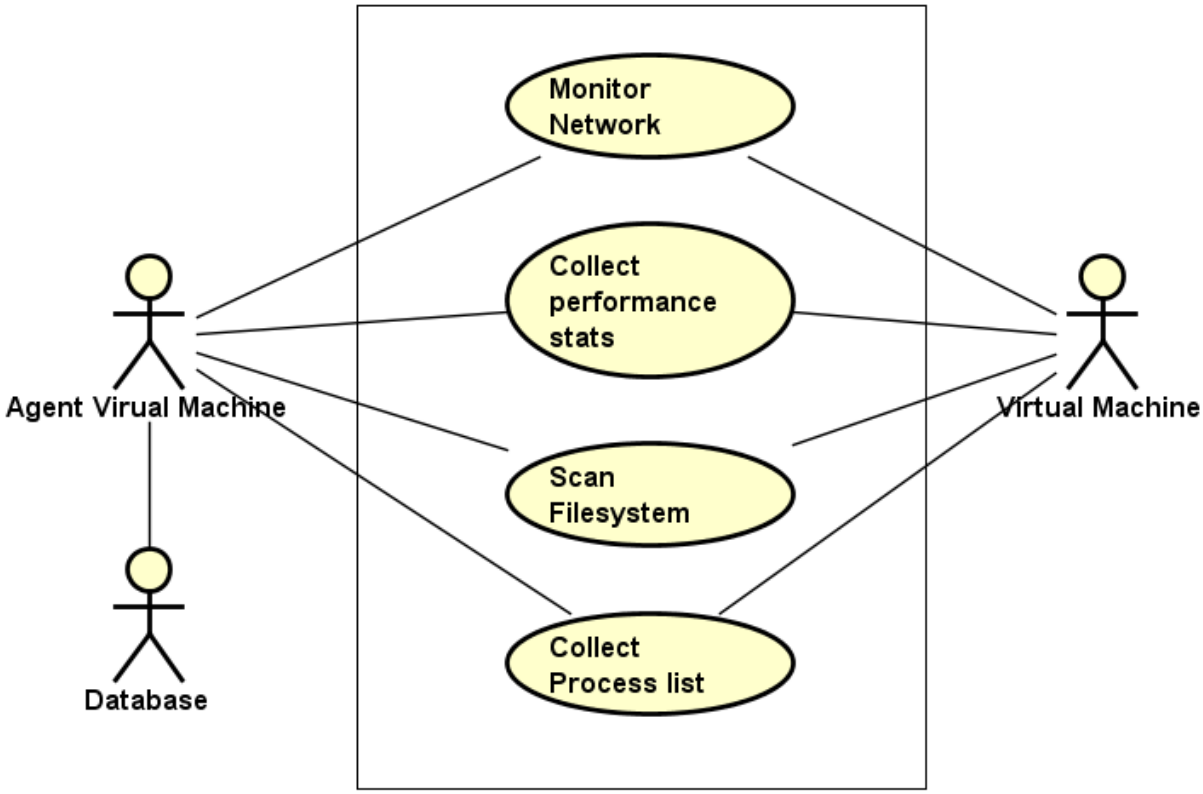


Figure 17 Use case diagram 2

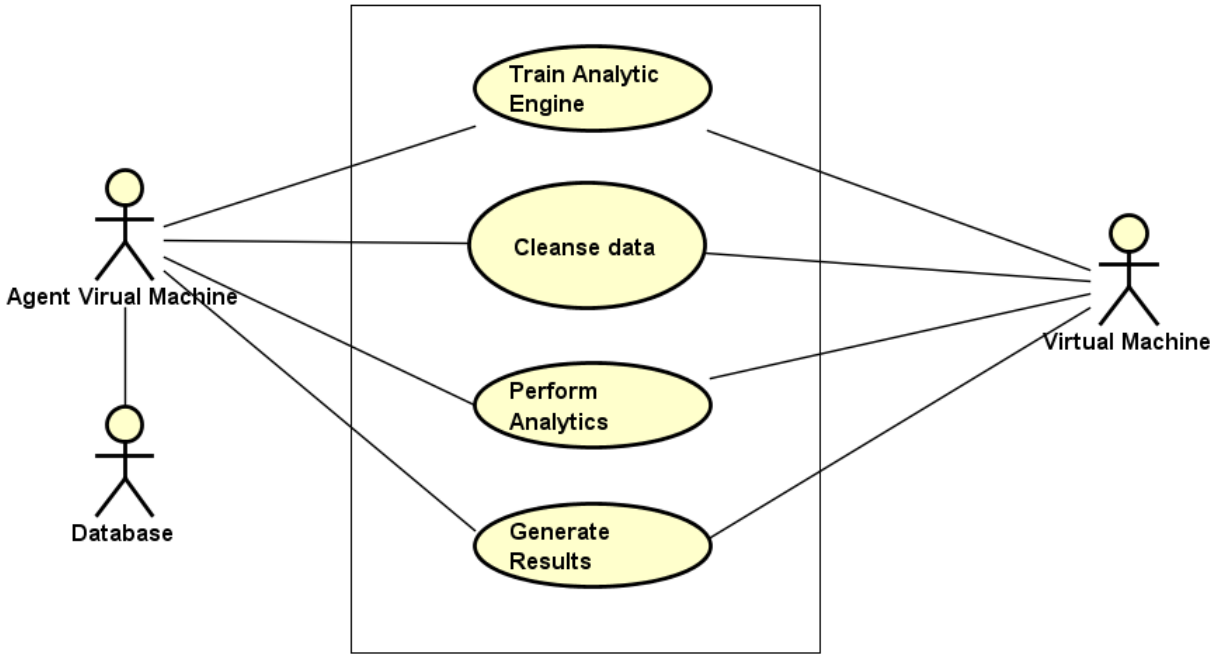


Figure 18 Use case diagram 3

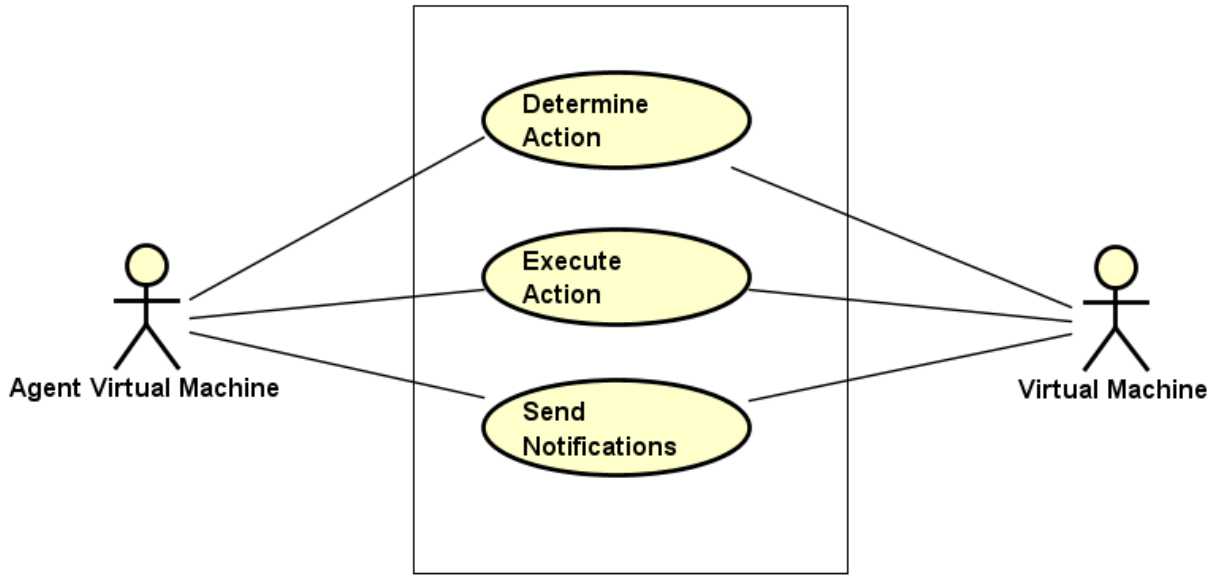


Figure 19 Use case diagram 4

Use Case description

Use Case: Establish Connection
ID: UC 01
Brief description: The Agent Virtual Machine establishes connection with the virtual machines in the network.
Primary Actors: Agent Virtual machine, Virtual machine
Secondary Actors: None
Pre-Conditions: The Agent Virtual machine is up and running.
Main flow: 1. Include (Authentication). 2. The Agent Virtual Machine collects the list of available virtual machine IPs in a network from the vCenter server. 3. The Agent VM initiates ping to Virtual machine to verify connectivity. 4. If the Virtual machine responds, the Agent VM confirms that the Virtual Machine is On. 5. The steps 3 and 4 will be repeated by the Agent VM for all the Virtual machines in the network.
Post-Conditions: 1. The Agent VM should have verified connectivity with all the virtual machines in a vCenter server.

Use Case: Register Virtual Machine
ID: UC 02
Brief description: The Virtual Machine registers itself with the Agent virtual machine.
Primary Actors: Agent Virtual Machine, Virtual Machine
Secondary Actors: None
Pre-Conditions: The connection between Agent Virtual machine and virtual machine is established.
Main flow: 1. The Agent initiates a register request to the Virtual machine. 2. The Virtual Machine authenticates the request before proceeding with the registration. 3. The Virtual Machine sends its Name, Operating System, Memory and other details to the Agent VM. 4. The Agent VM creates a unique record in its database to register the actual VM details. 5. The Agent VM sends an acknowledgement to the actual VM once the registration is successful. 6. The actual VM decodes the error code from the Agent VM response (acknowledgement).
Rainy day Scenario: 1. If the actual VM is unable to register its details, the request will be retired. 2. Steps 1 to 6 will be repeated.
Post-Conditions: 1. The actual VM is registered successfully with the Agent VM.

Use Case: Authentication
ID: UC 03
Brief description: The Agent Machine authenticates its credentials with vCenter to establish a session.
Primary Actors: Agent Virtual Machine, Virtual Machine
Secondary Actors: None
Pre-Conditions: The administrator should have necessary credentials to access VMs in vCenter.
Main flow: 1. The administrator provides vCenter IP address, username and password to connect to vCenter 2. The Agent machine initiates a request to authenticate the user. 3. The Agent machine invokes the session web service in vCenter to authenticate the credentials. 4. If authentication is successful, session will be established to process further requests. 5. If authentication fails, error message will be send back to Agent machine.
Post-Conditions: 1. The Agent VM authenticates its connection with vCenter server.

Use Case: Collect Network data
ID: UC 04
Brief description: The Virtual machine network data will be monitored by the Agent VM, to check if the machine is sending any data to machines outside the network.

Primary Actors: Agent virtual machine, Virtual Machine
Secondary Actors: None
Pre-Conditions: The virtual machine is up and running.
Main flow: 1. The Agent VM initiates a request to collect network statistics from actual VM. 2. The virtual machine authenticates the request from the Agent VM. 3. The virtual machine executes the necessary commands to collect necessary information. 4. The virtual machine generates the output that is readable by the Agent VM. 5. The virtual machine sends the collected data as a response to the Agent VM. 6. The agent VM inserts the data into the database. 7. The Agent VM sends an acknowledgement to the virtual machine.
Post-conditions: The network statistics from the virtual machine is collected successfully.

Use Case: Collect Performance data
ID: UC 05
Brief description: The Virtual machine network data will be monitored by the Agent VM, to identify recurring patterns in the machine's performance. The performance statistics data like memory, CPU usage etc will be send to the Agent VM.
Primary Actors: Agent virtual machine, Virtual Machine
Secondary Actors: None

Pre-Conditions: The performance counters of the virtual machine is accessible.
Main flow: 1. The Agent VM initiates a performance collect request from actual virtual machine. 2. The actual VM authenticates the request. 3. The virtual machine executes an internal service call to collect the performance measure values. 4. The virtual machine puts the data in a JSON format. 5. The virtual machine repeats step 2 to 4 if the performance data has to be collected for a specific time period. 6. The virtual machine then sends the response back to Agent VM. 7. The Agent VM inserts the data into the database and sends an acknowledgement to the virtual machine.
Post-conditions: The performance statistics from the virtual machine is collected successfully.

Use Case: Determine and Execute Action
ID: UC 06
Brief description: The Agent Virtual Machine determines action for the malware prevention in the network.
Primary Actors: Agent Virtual machine, Detection System
Secondary Actors: None
Pre-Conditions: The Agent Virtual machine is up and running and the network is in sync.

Main flow: 1. The Agent machine collects the Malware analysis data from the detection system. 2. The Agent machine reads the probability value from the analyzed data. 3. The Agent machine alerts the administrator about the malware existence in the virtual machine. 4. If the probability value is too high, the Agent VM quarantines the affected VM and isolates it from the network. 5. The Agent machine updates the User Interface with the latest status of Virtual machine. 6. The Agent machine updates the VM status information in the MongoDB instance.
Post-Conditions: 1. The Agent machine must have taken necessary action on the affected VM.

Use Case: Scan files
ID: UC 07
Brief description: The Agent Virtual Machine scans all the files in the system of the virtual machines in the network.
Primary Actors: Agent Virtual machine, Virtual machine
Secondary Actors: None
Pre-Conditions: The Agent Virtual machine is up and running.
Main flow: 1. Include (Authentication). 2. The Agent Virtual Machine collects the list of available virtual machine IPs in a network

from the data center. 3. The Agent VM makes sure that the systems are in the network. 4. The Virtual machine scans the files of all the systems in the network. 5. The Agent Virtual machine will scan and checks for the unwanted files present in the system. 6. The steps 3 to 5 will be repeated by the Agent VM for all the Virtual machines in the network.
Post-Conditions: 1. The Agent VM should have access to check the file system and to monitor the unwanted files on the network.

Use Case: Monitor processes
ID: UC 08
Brief description: The Agent Virtual Machine checks the Process List of the virtual machines in the network.
Primary Actors: Agent Virtual machine, Virtual machine
Secondary Actors: None
Pre-Conditions: The Agent Virtual machine is up and running.
Main flow: 1. Include (Authentication). 2. The Agent Virtual Machine collects the list of available virtual machine IPs in a network from the data center. 3. The Agent VM makes sure that the systems are in the network.

4. The Virtual machine scans the systems in the network and get the processes running on the system. 5. The Agent Virtual machine monitor the processes and will see for any new and suspicious processed running in the system. 6. The steps 3 to 5 will be repeated by the Agent VM for all the Virtual machines in the network.
Post-Conditions: 1. The Agent VM should have data of the processes running on the system and hence can detect any malware present in the network.

Use Case: Training Analytic Engine
ID: UC 09
Brief description: The Analytic engine which is build needs to be trained for initial prediction.
Primary Actors: Agent Virtual machine, Analytic Engine.
Secondary Actors: None
Pre-Conditions: The sample data set is collected and Analytic engine is built.
Main flow: *1.* The Agent virtual machine triggers the analytic engine. *2.* The analytic engine requests for training data set from Agent virtual machine. 3. The Agent machine fetches training dataset from database to analytic engine. 4. The analytic engine cleanses the data and calls machine learning libraries to process training data set. 5. The analytic engine uses test data to verify the accuracy of result. 6. The analytic engine sends the resulting data to agent machine. 7. Repeat steps 2 to 6 for training analytic engine for improving accuracy of the result.
Post-Conditions: 1. The Analytic engine must result in desired level of accuracy.

Use Case: Performing Analytics
ID: UC 10
Brief description: The Analytic engine which is build needs to perform analytics on actual data to generate results.
Primary Actors: Agent Virtual machine, Analytic Engine.
Secondary Actors: None
Pre-Conditions: Analytic engine is built and cleaned data sets are ready.
Main flow: *1.* The Agent virtual machine triggers the trained analytic engine. 2. The Analytic engine request for cleaned malware data set from Agent Machine. 3. The Agent machine feeds the collected data from database to Analytic engine 4. The Analytic engine calls machine learning libraries to perform analytics. 5. The libraries uses Machine learning classifier algorithms to analyse the data. 6. The Generated output is send back to agent machine.
Post-Conditions: 1. The output file must be generated in defined format.

Sequence diagram

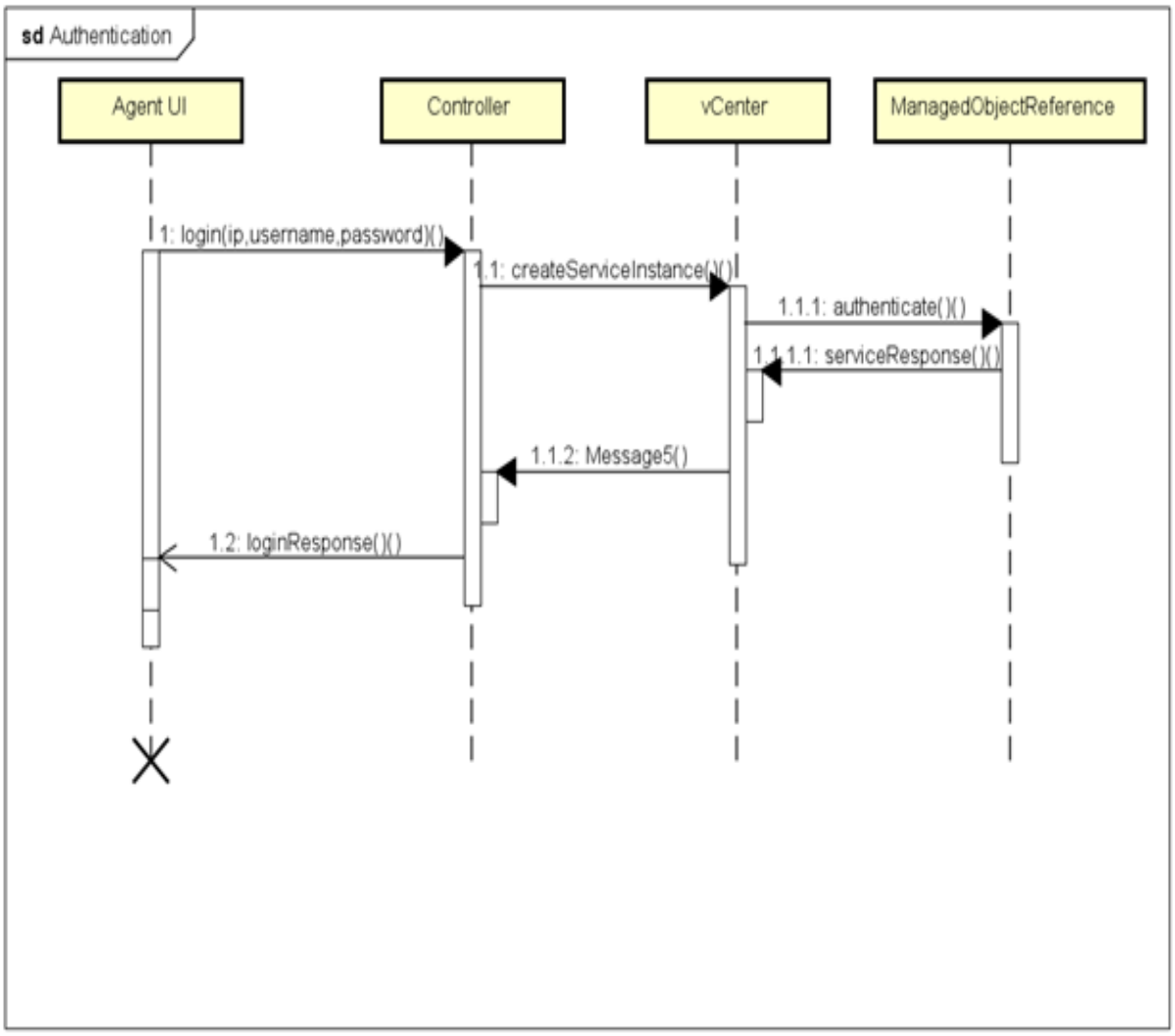


Figure 20 Sequence diagram - Authentication

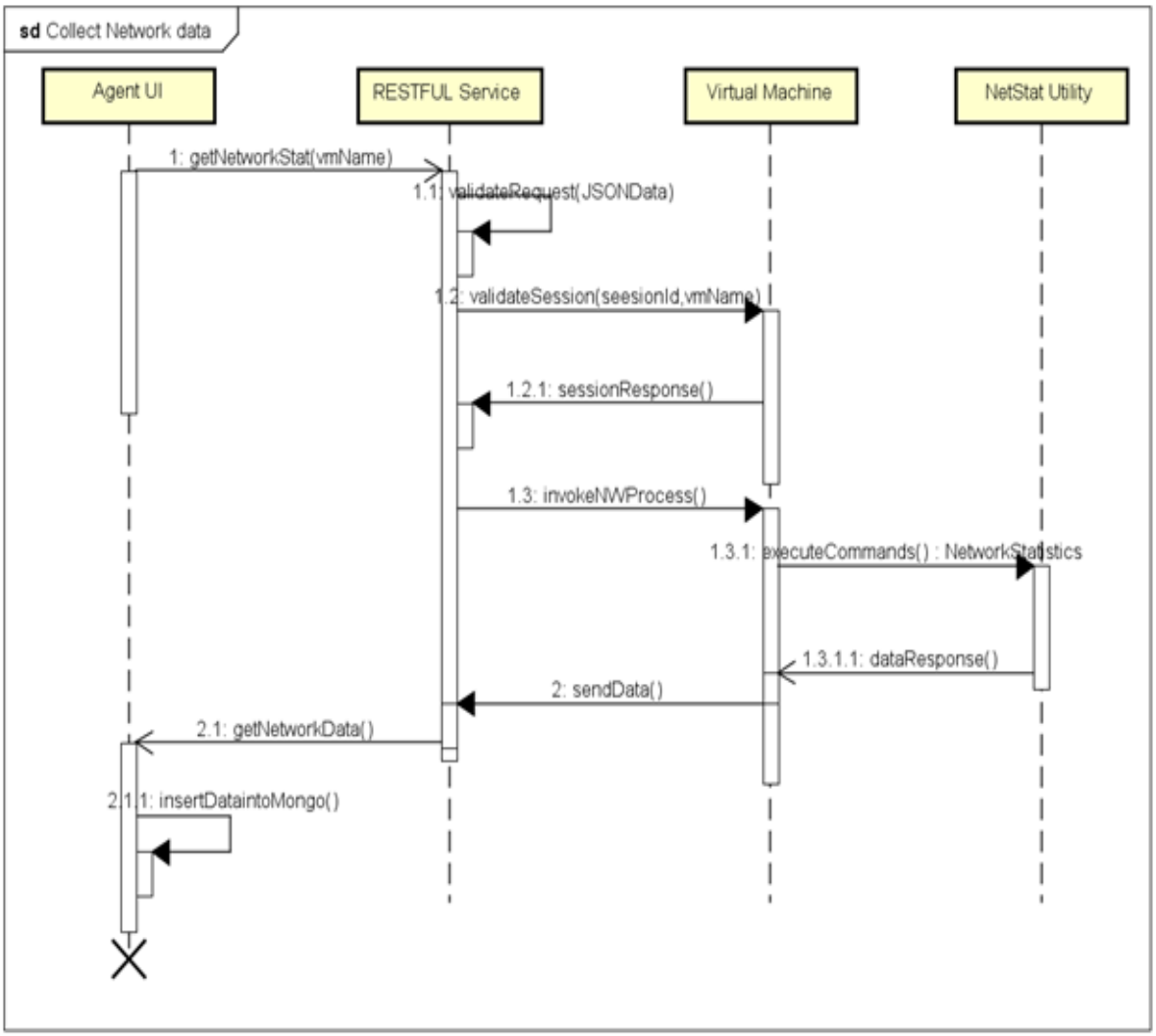


Figure 21 Sequence diagram - Collect network data

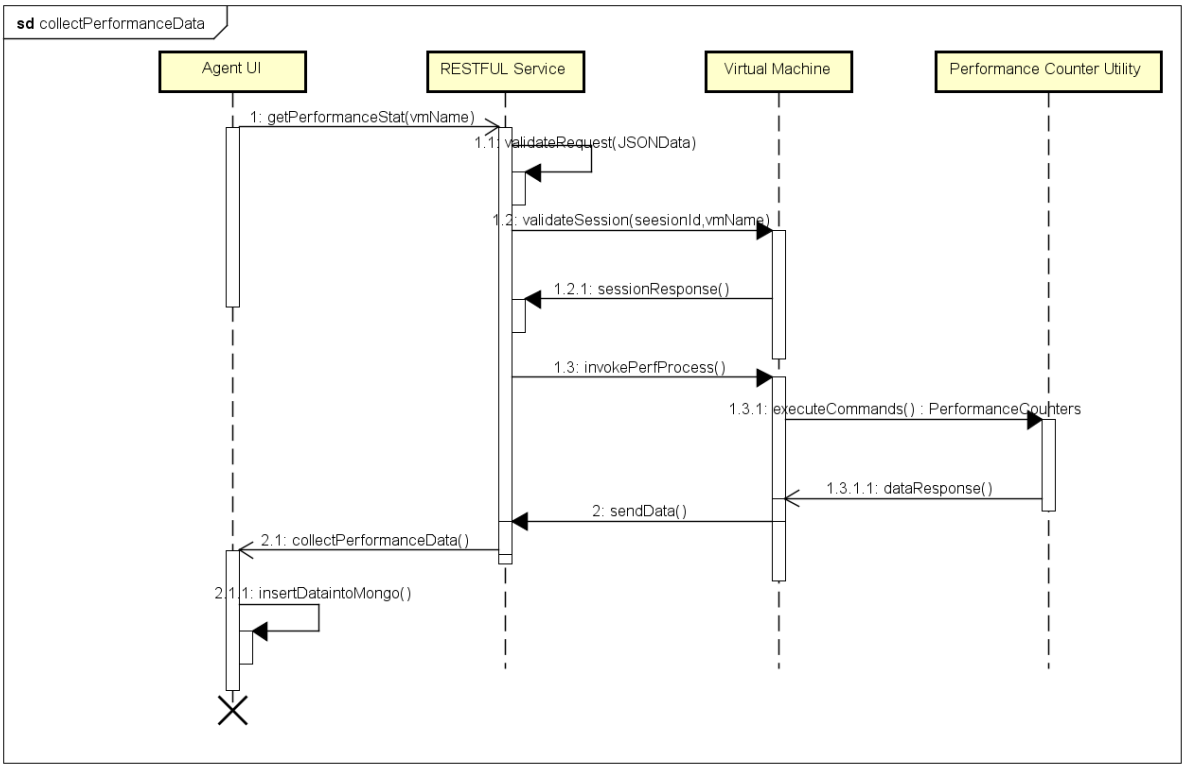


Figure 22 Sequence diagram - Collect network data

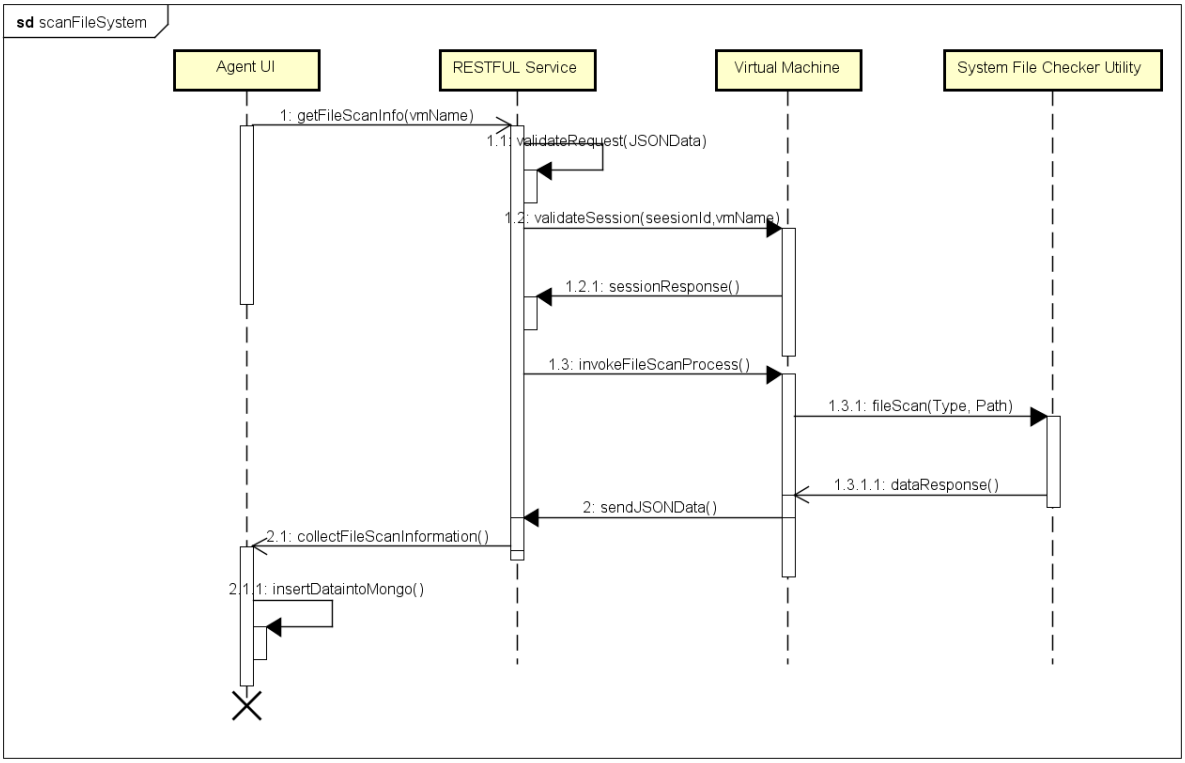


Figure 23 Sequence diagram - Scan file system

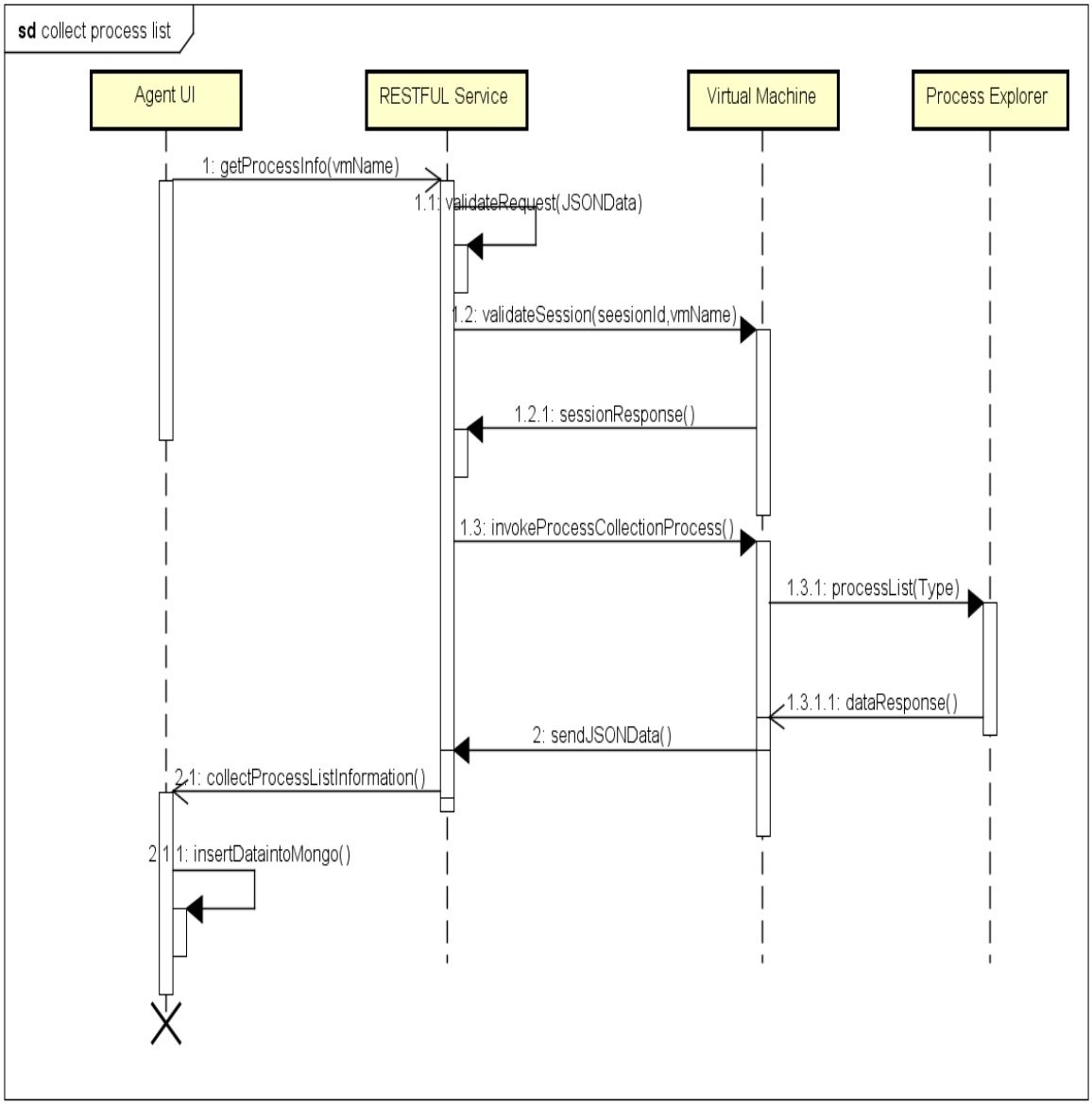


Figure 24 Sequence diagram - Collect process list

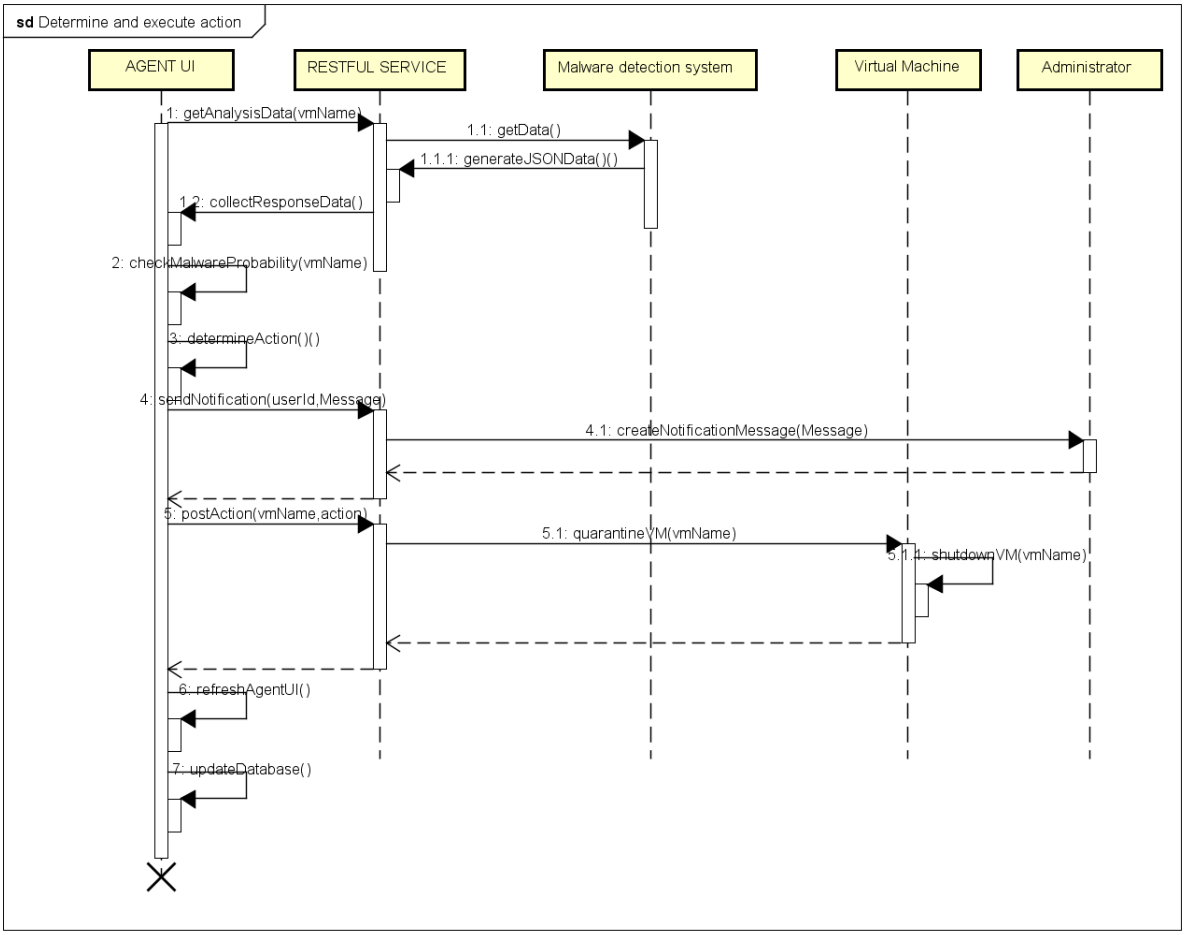


Figure 25 Sequence diagram - Determine and execute action

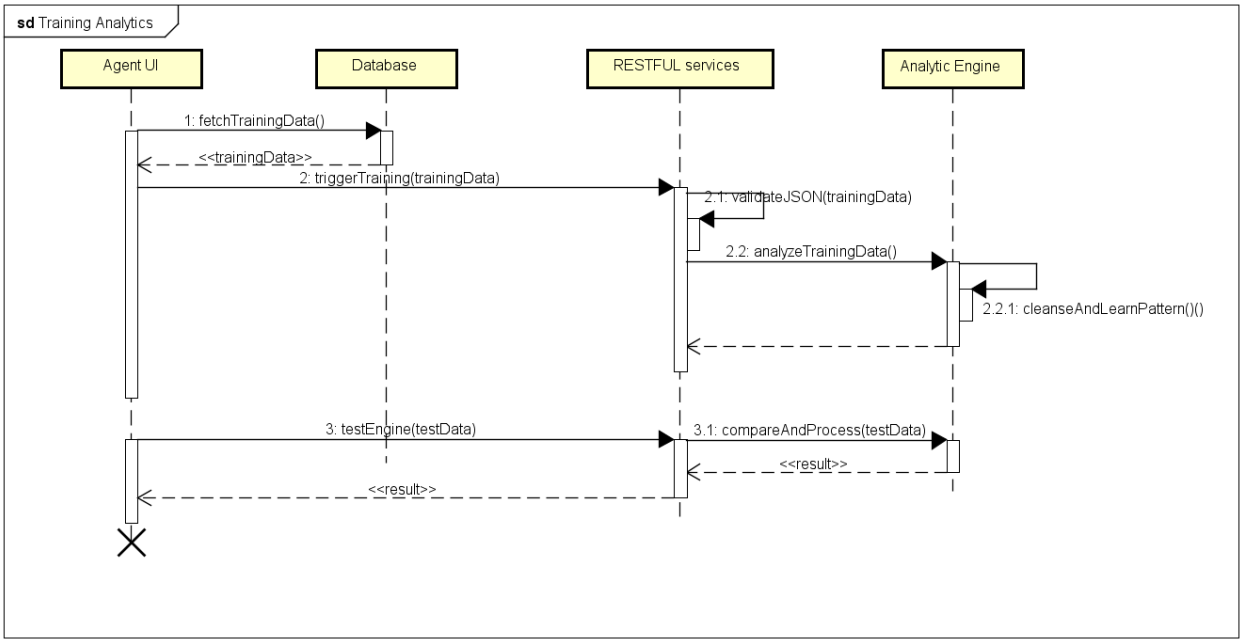


Figure 26 Sequence diagram - Training Analytics Engine

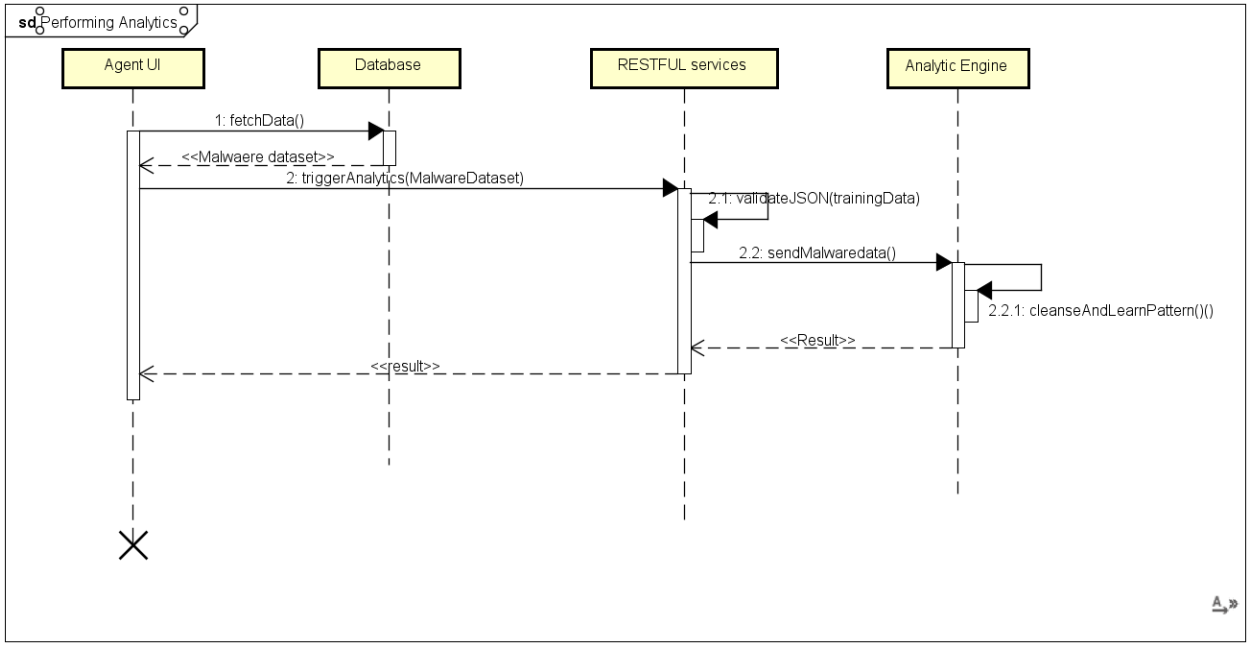


Figure 27 Sequence diagram - Performing Analytics

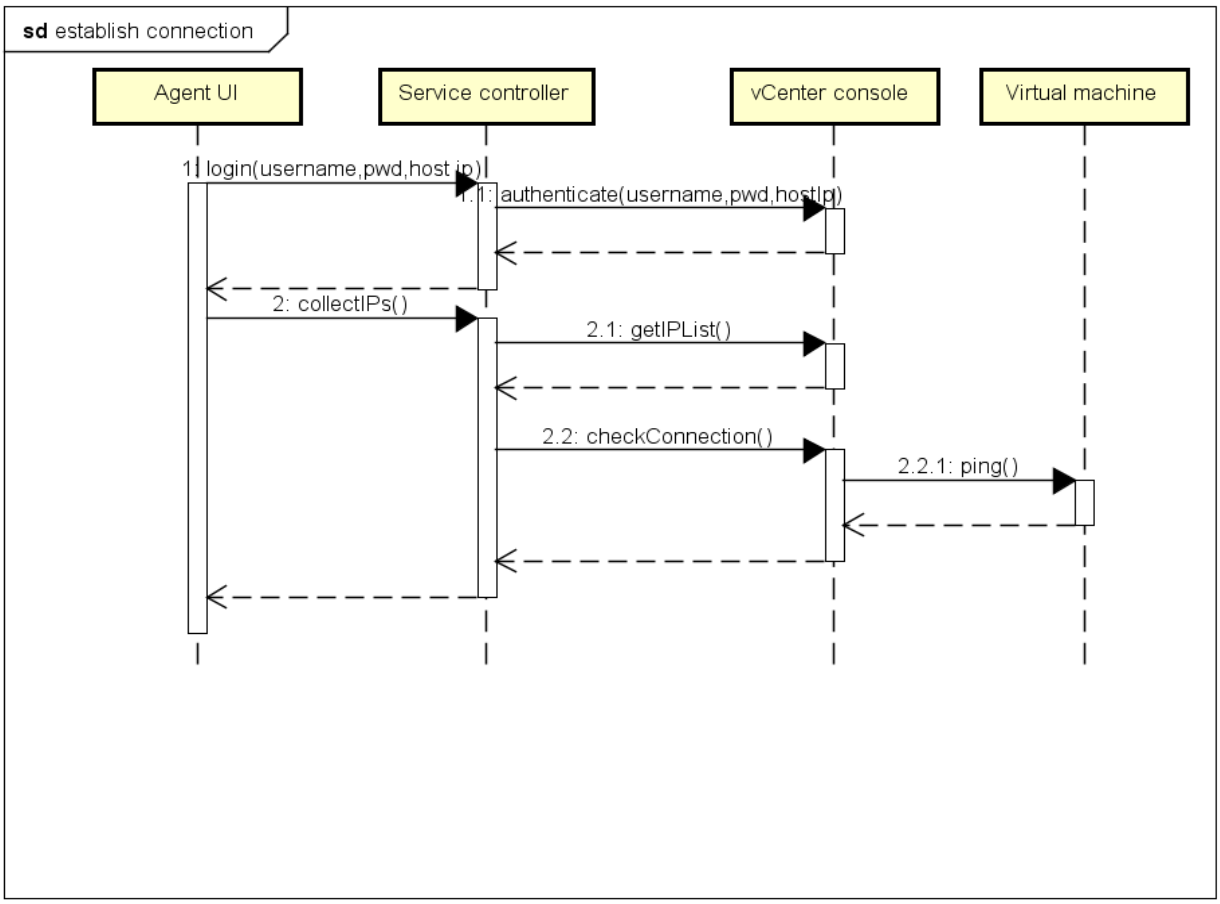


Figure 28 Sequence diagram - Establish Connection

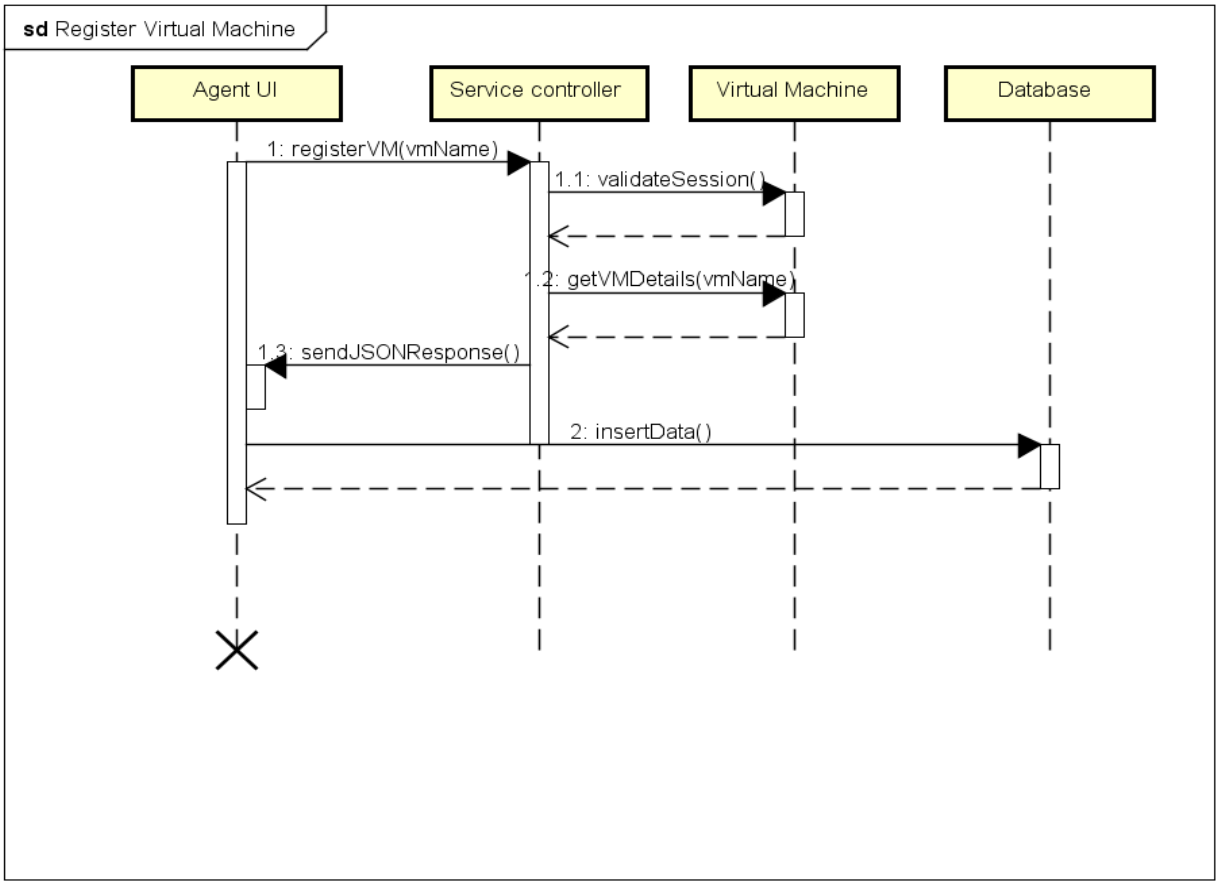


Figure 29 Sequence diagram - Register virtual machine

Chapter 5 Project Implementation

Client Implementation

Following are the screenshots of the flow of the project.

1. Login Module:

Admin is required to login to the application. Admin will have to scan the VMs in the cloud network.

Admin have to sign up in the application and create its account for logging into the system.

Admin Validation is also handled in the application.

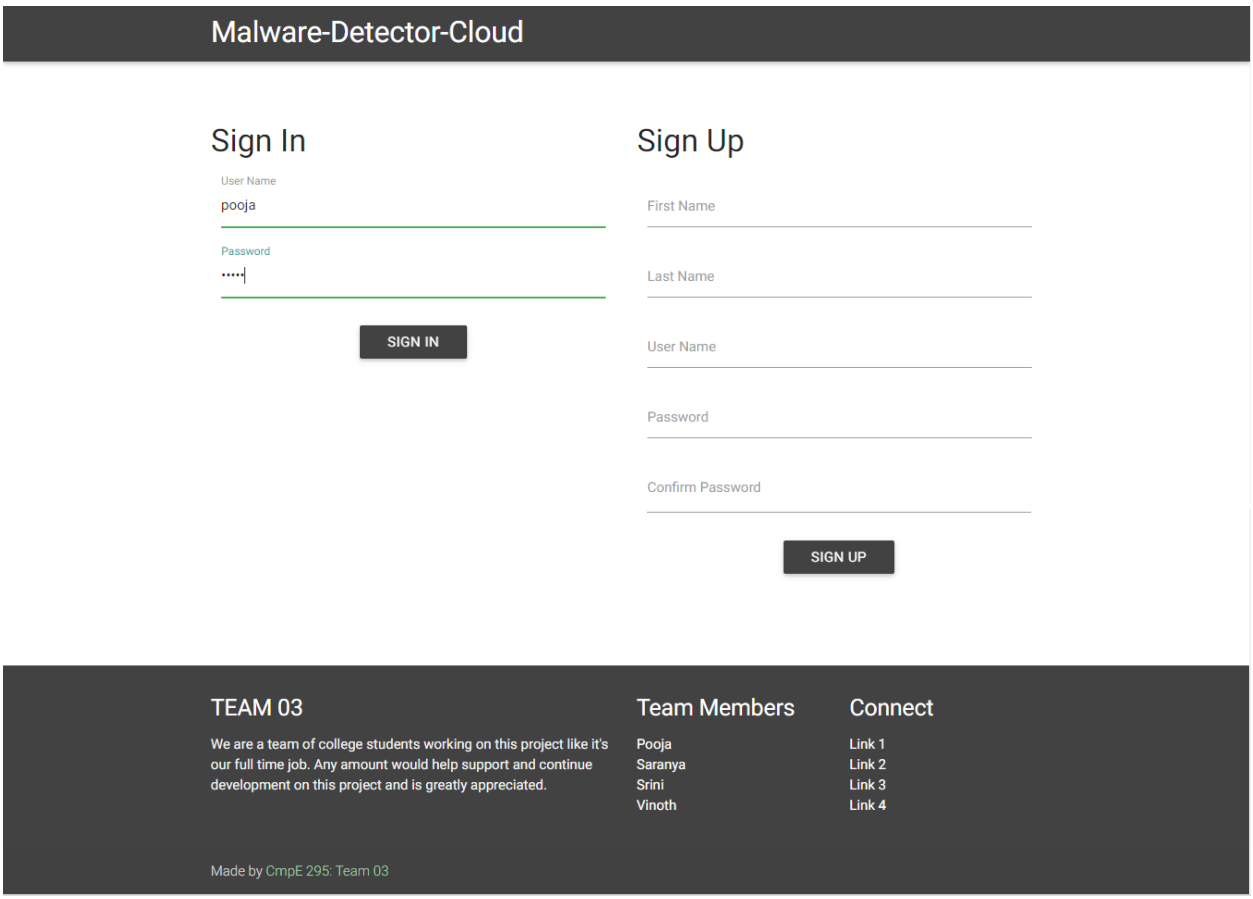
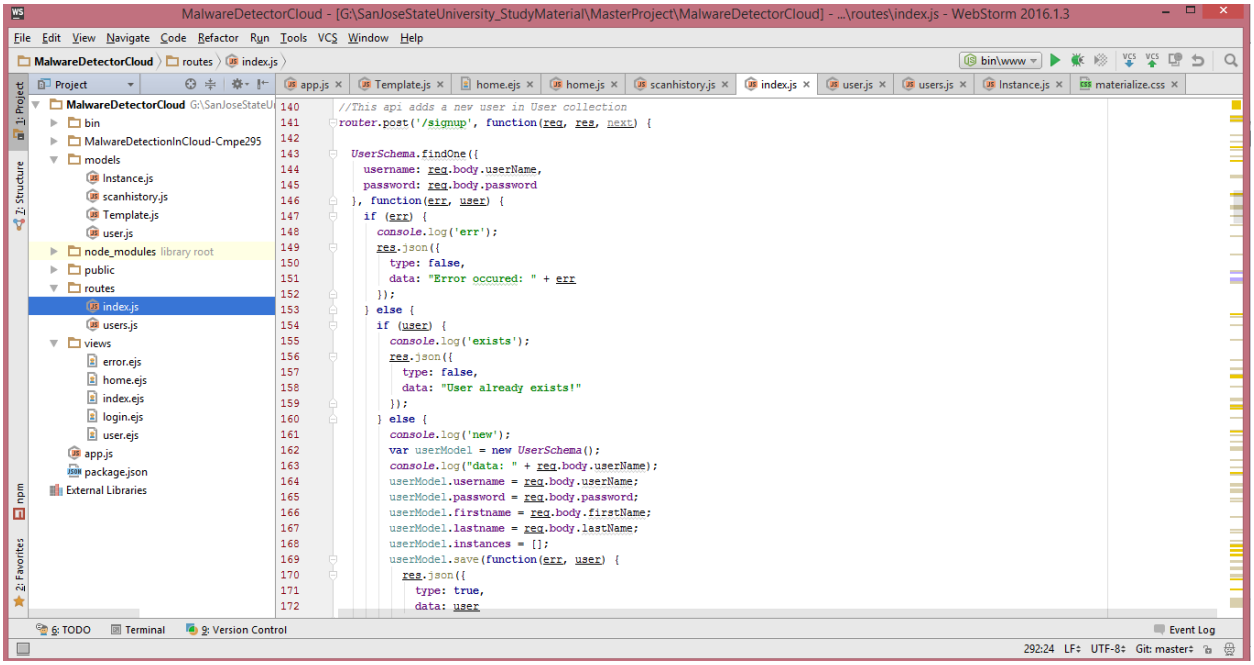
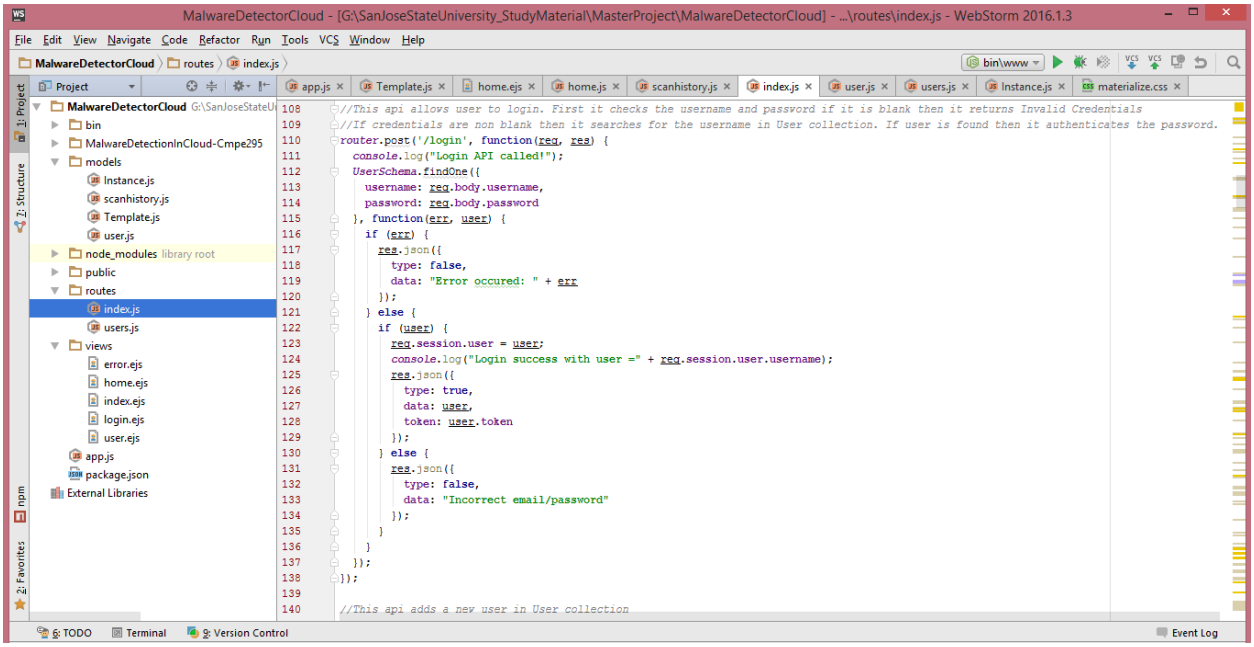


Figure 30 Screenshot for Sign-in and Sign-up page




```
//This api adds a new user in User collection
router.post('/signup', function(req, res, next) {

  UserSchema.findOne({
    username: req.body.userName,
    password: req.body.password
  }, function(err, user) {
    if (err) {
      console.log('err');
      res.json({
        type: false,
        data: "Error occured: " + err
      });
    } else {
      if (user) {
        console.log('exists');
        res.json({
          type: false,
          data: "User already exists!"
        });
      } else {
        console.log('new');
        var userModel = new UserSchema();
        console.log("data: " + req.body.userName);
        userModel.username = req.body.userName;
        userModel.password = req.body.password;
        userModel.firstname = req.body.firstName;
        userModel.lastname = req.body.lastName;
        userModel.instances = [];
        userModel.save(function(err, user) {
          res.json({
            type: true,
            data: user
```




```
//This api allows user to login. First it checks the username and password if it is blank then it returns Invalid Credentials
//If credentials are non blank then it searches for the username in User collection. If user is found then it authenticates the password.
router.post('/login', function(req, res) {
  console.log("Login API called!");
  UserSchema.findOne({
    username: req.body.username,
    password: req.body.password
  }, function(err, user) {
    if (err) {
      res.json({
        type: false,
        data: "Error occured: " + err
      });
    } else {
      if (user) {
        req.session.user = user;
        console.log("Login success with user =" + req.session.user.username);
        res.json({
          type: true,
          data: user,
          token: user.token
        });
      } else {
        res.json({
          type: false,
          data: "Incorrect email/password"
        });
      }
    }
  });
});

//This api adds a new user in User collection
```

Figure 31 Implementation for Sign-in and Sign-up page

After signing up and loging into the system the admin can see the dashboard and can see the listing of VMs,scan history,statistics and even launch the instance in the network.

Admin can monitor the VMs in the cloud network.

Vms are scanned and the scan resuts are displayed and the scan behviour in the dashboard.

We can stop the VM if we find any malicious program in the network.

2. Dashboard Module :

1. Screenshot for Dashboard – Listing of all the VMs in the network.
   VM Name, OS type, Ip address of the Machine and Status of the Machine is displayed on the screen. The Vms which are in the Cloud network which are seen in the Vsphere Client UI are fetched to displa on the screen.

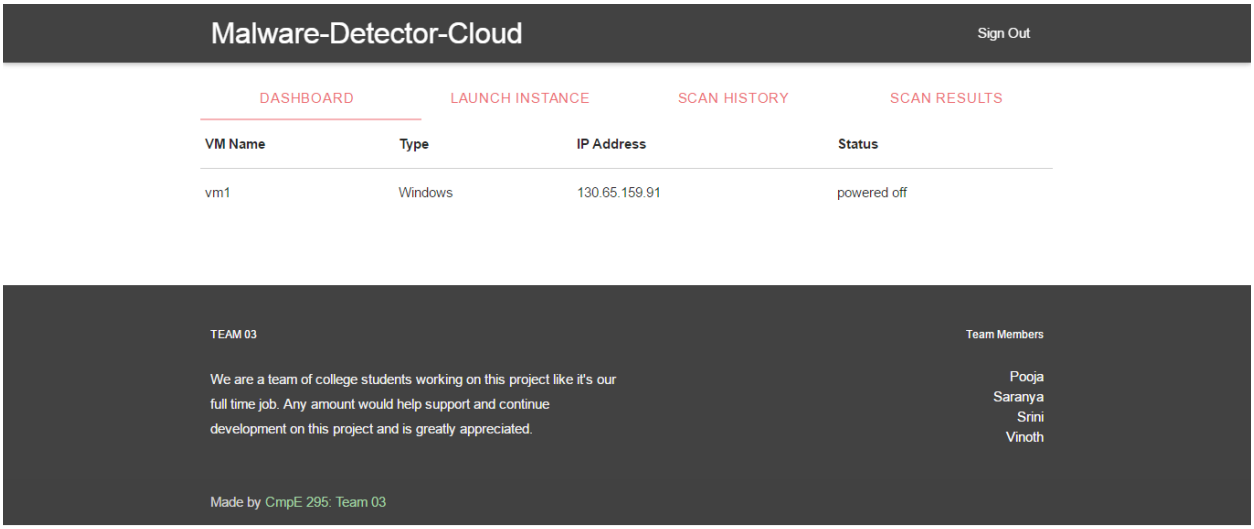


Figure 32 Screenshot for Dashboard page

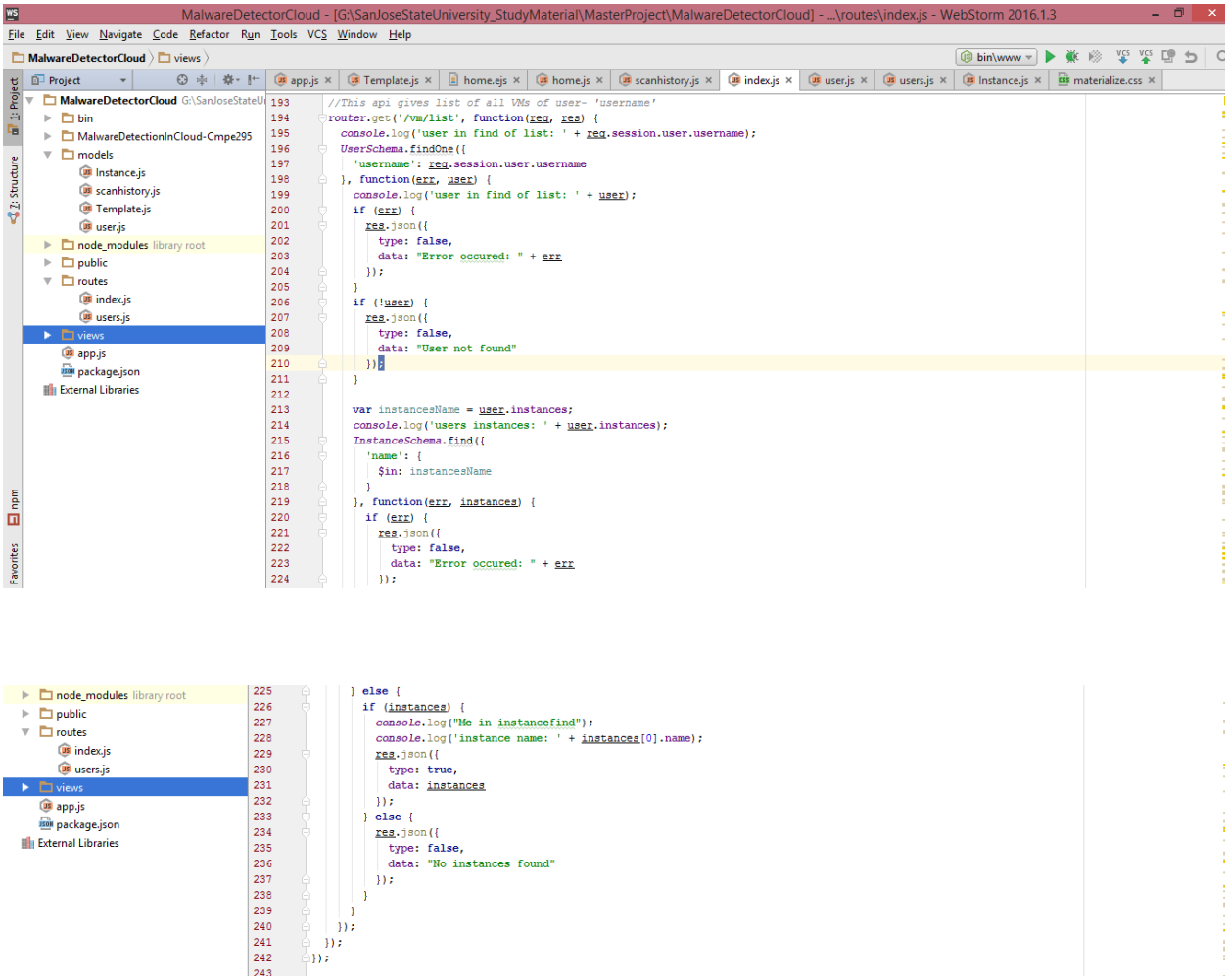


Figure 33 Implementation for Dashboard page

3. Creation/Launching of VM:

2. Screenshot for Launching of the VM.

We can launch the machine in the network. The templates for Ubuntu, Windows are available and we create the VM from the template.

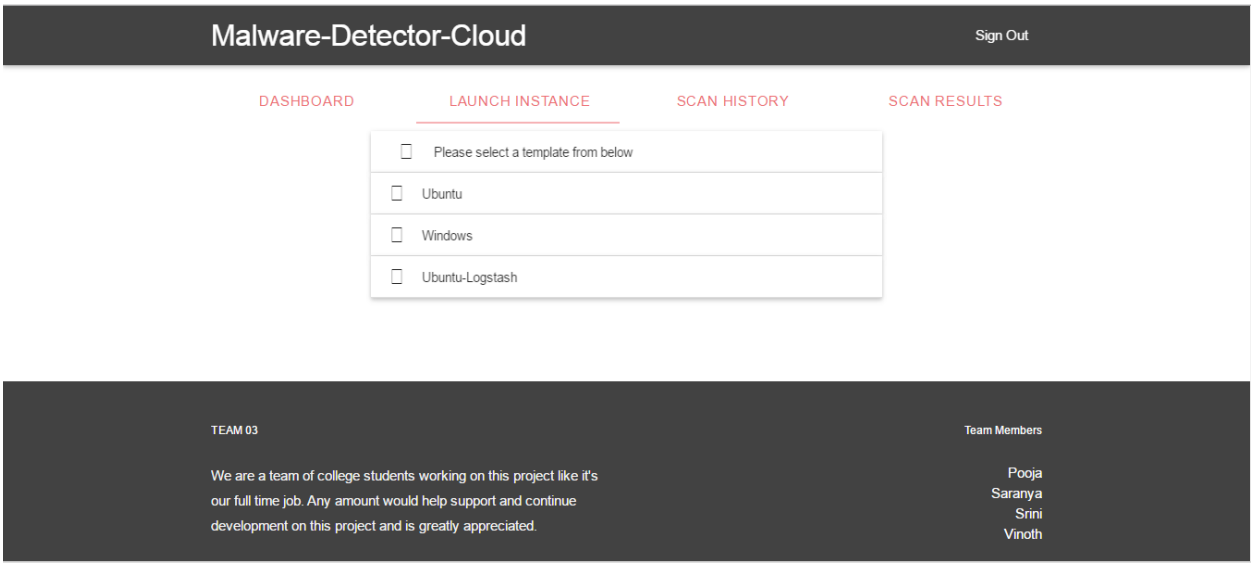


Figure 34 Screenshot for creation on VM page

4. VM Scan Module:

3. Screenshot for the Scan history for all the VM.
   After scanning the VM, the algorithm gets the scan results to detect any malicious program in the VM. The scan id, Status, VM name, Verbose Message and File name is populated for the respective machine.

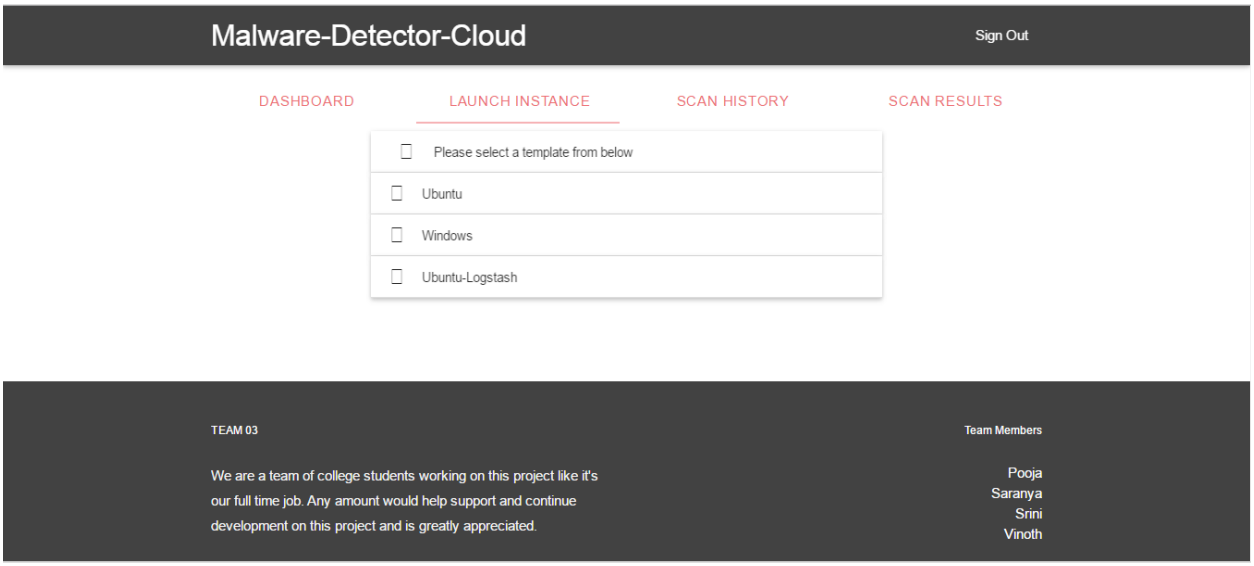


Figure 35 Screenshot for VM details

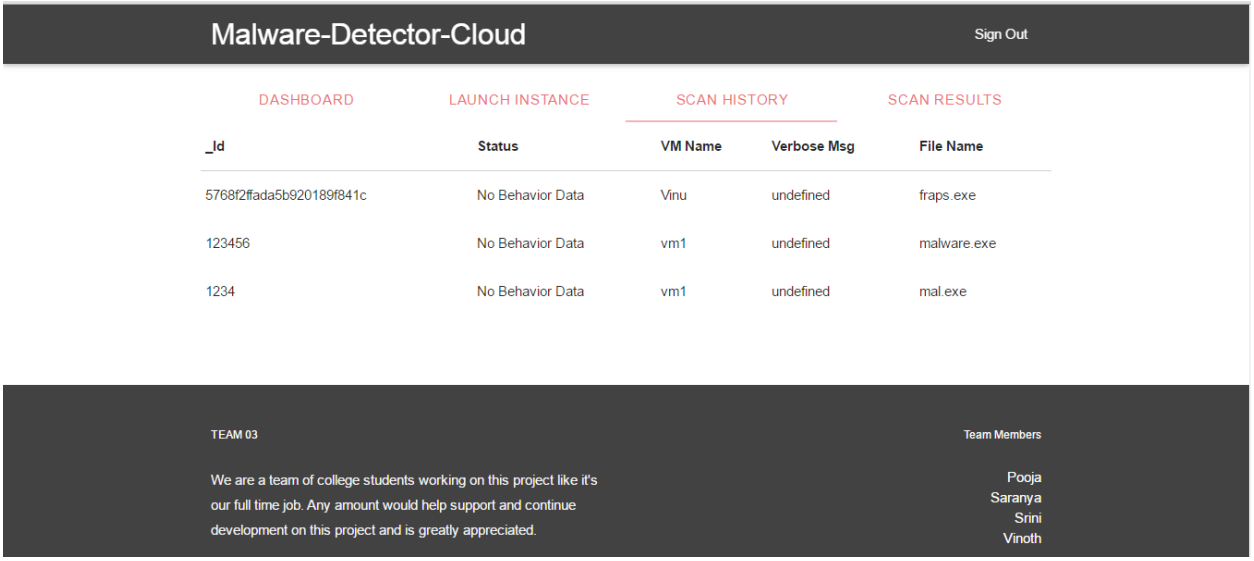


Figure 36 Screenshot for active VMs dashboard

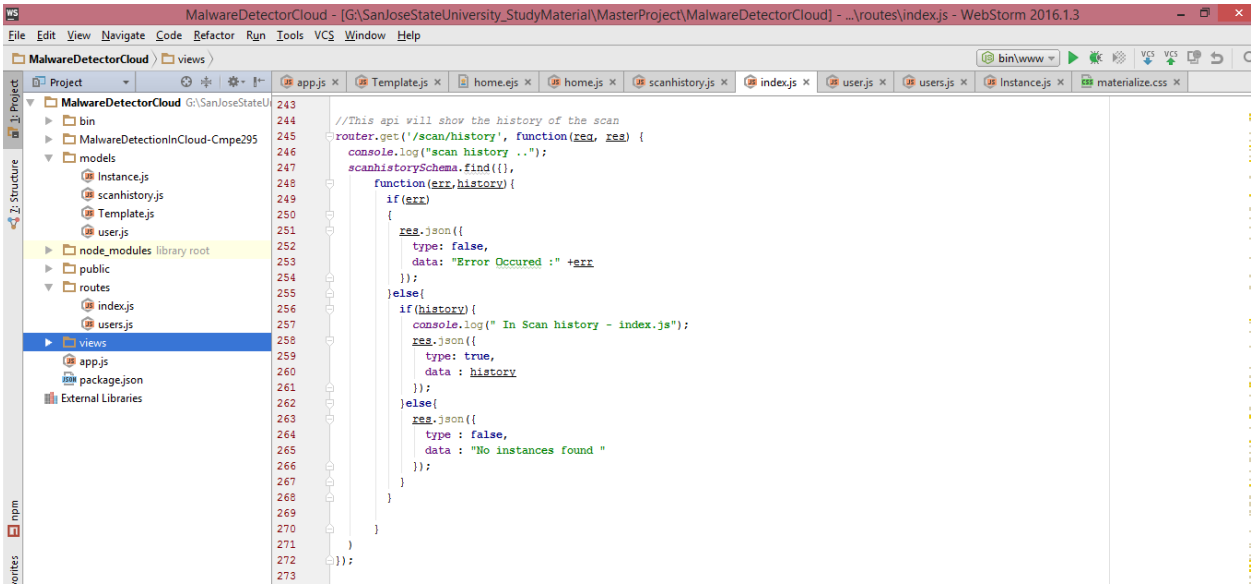


Figure 37 Implementation for VM Dashboard page

4. Screenshot for the VM Details Page.

We can have the details of the VM on the VM Detail page.

Following are the details we can see on the VM Details page – VM Name, OS name, Memory and CPU of the VM. Status of the VM, Ip address of the Machine.

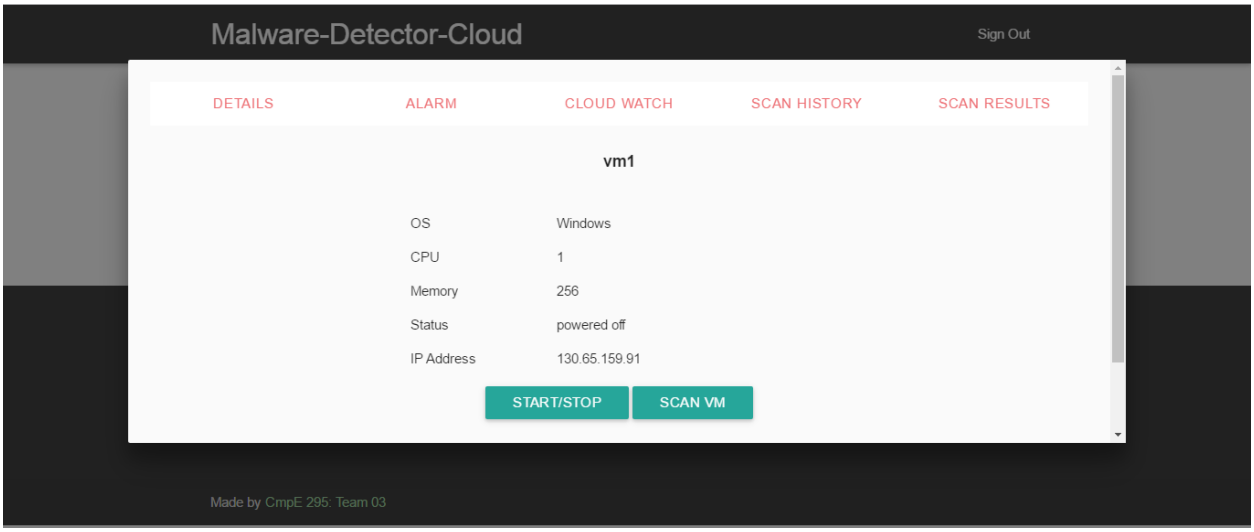


Figure 38 Screenshot for active VMs dashboard

5. Screenshot for the Scan History for the Particular VM.
   Scanning is performed for the VMs in the network. All the scan history is saved in the Mongo DB for the VMs. The scan history is available for the particular VM and all the VMs together. We get to see the scan history for respective scan of the VM.

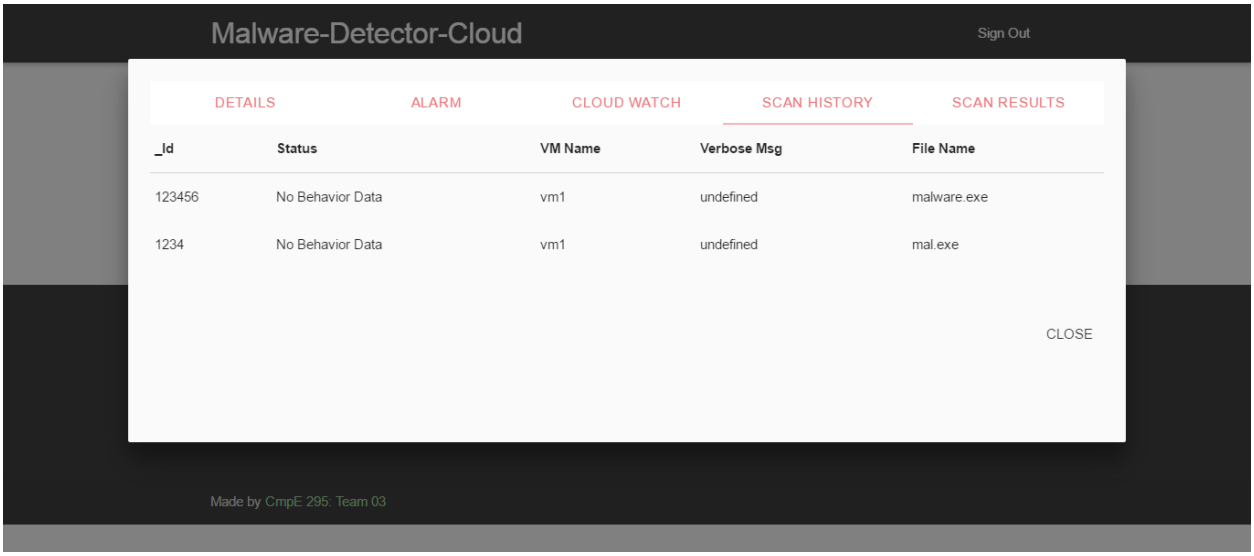


Figure 39 Screenshot for active VMs behavioral Data

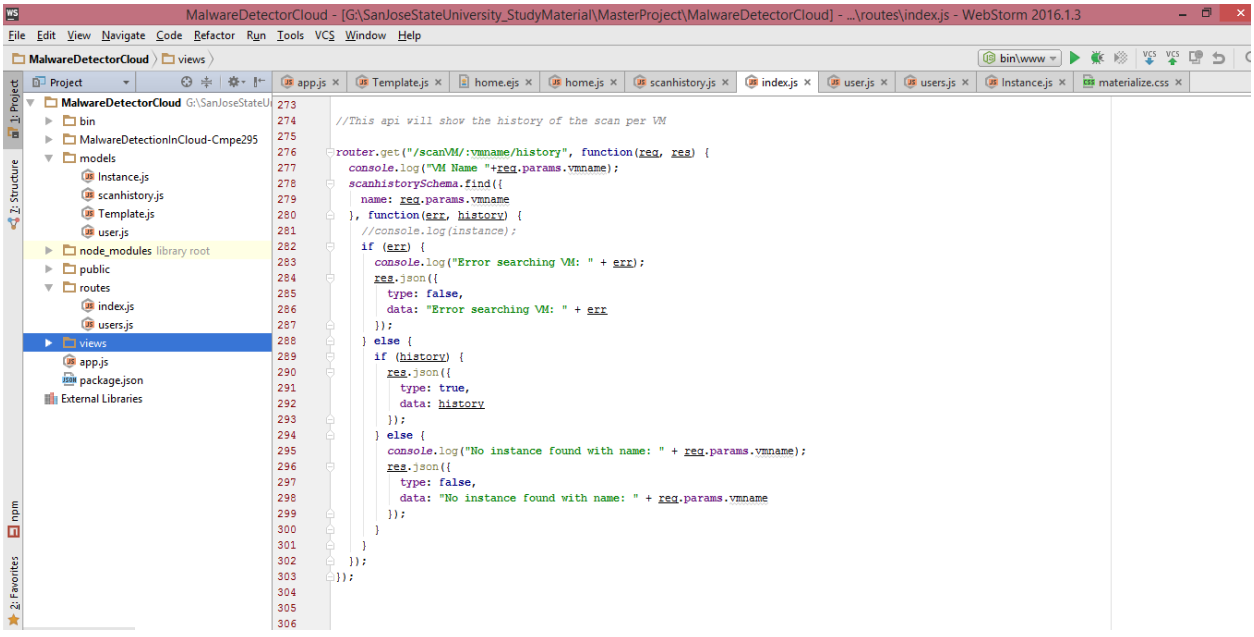


Figure 40 Implementation for active VMs dashboard

5. VSphere Module:

6. Screenshot for the VSphere Client.

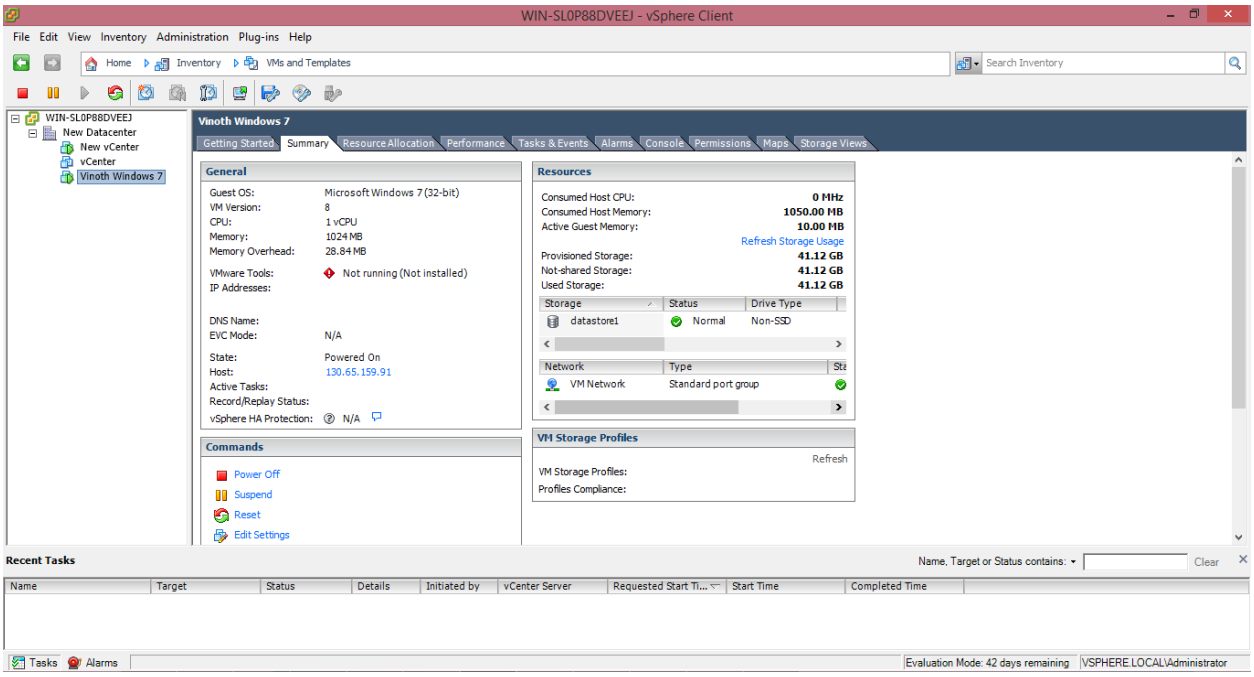


Figure 41 Screenshot of VSphere Client

6. Statistics and Graphs Module:

7. Screenshot for the Graphs – CPU and Memory.

   Using high Charts, Graphs are generated at the runtime and we can see the latest graphs depending on the metrics and statistics of the VM. CPU usage, performance graph, Memory usage, Network Graph for the VM is populated on the screen for a particular VM selected.

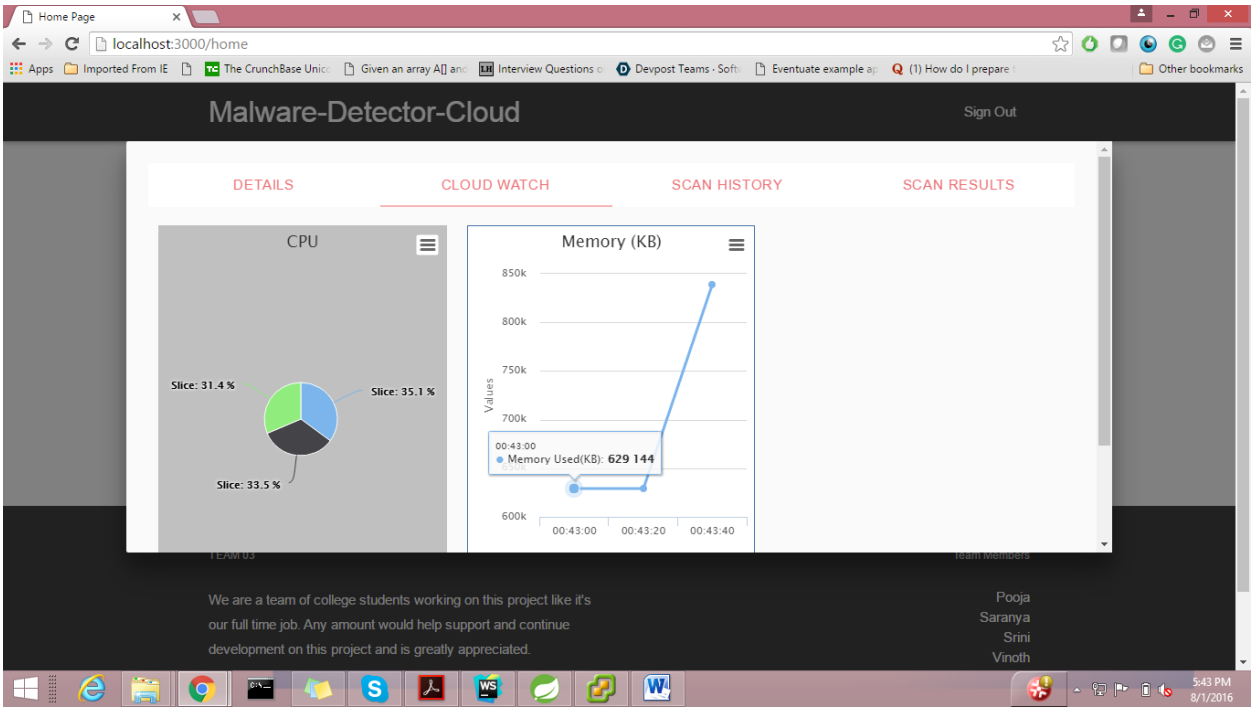


Figure 42 Screenshot of VM stats graphs

Middle Tier Implementation

A middle tier application plays an important role in integrating the front end and the backend. But in the context of our application, the middle tier plays additional roles of scanning the malwares from individual virtual machines and to orchestrate the malware analysis. The detailed list of malware responsibilities will be discussed in detail.

The administrator initiates a scan request from the web application. The middle tier handles the request and performs a scan on the virtual machine. The virtual machine executes each process in a sandboxed environment. The analyzer provides a behavioral information of each process from the sandbox environment. The data is then pushed to a central database. The malware analysis module uses machine learning algorithm to analyze the data.

Responsibilities

The following are the application responsibilities of middle tier.

- Handle scan requests from User Interface
- Validate scan requests from virtual machines that are powered on
- Perform process watcher i.e. to identify the executables that are newly created. For the scope of our implementation, the executables present in prominent malware directories will be watched.
- Initiate requests to execute the malwares in a sandboxed environment.
- Generates and stores unique scan ids in a NoSQL database.
- Initiates a RESTful call to get the behavior data from the Sandbox analyzer. Behavior data includes network, process, memory, registry and API calls made by each executable file.
- Generate Feature vector necessary for the Machine learning algorithm in a CSV format.
- Store and maintain scan history and behavior data of both malicious and benign dataset.

Components

Following are the core components of our application middle tier. Each component and its functionalities are discussed in the subsequent chapters.

- Scan request handler
- Malware scanner
- VirusTotal Sandbox analyzer
- Behavior data extraction engine
- Feature Vector generation module
- Malware Analyzer (Machine learning)
- Data store and data management

Scan request handler

The scan request handler consists of services that accepts request from web application. The scan request handler initiates the scan request to scan the executable present in malware prone directories in a virtual machine. VIX API from VMWare will be used to invoke program execution on a virtual machine from the user interface. In order to execute the program

remotely, VMWare tools should be installed in the virtual machine to accept VIX commands. VIX used GuestOperationsManager class to send machine credentials and program specification to run the program. The following code is a snippet for scan request handler component.

```
public class ProgramRun
{
    public static void triggerProgram(String vmName){
    ServiceInstance si = new ServiceInstance(new URL("https://130.65.159.90/sdk"), "administrator@vsphere.local", "******", true);
    Folder rootFolder = si.getRootFolder();
    ManagedEntity[] mes = new InventoryNavigator(rootFolder).searchManagedEntities("VirtualMachine",vmName);
    if(mes==null || mes.length ==0)
    {
      return;
    }
    GuestOperationsManager gom = si.getGuestOperationsManager();

    VirtualMachine vm = (VirtualMachine) mes[0];
    GuestAuthManager gam = gom.getAuthManager(vm);
    NamePasswordAuthentication npa = new NamePasswordAuthentication();
    npa.username = "windows";
    npa.password = "testvm";

    GuestProgramSpec spec = new GuestProgramSpec();
    spec.programPath = "C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe";
    spec.arguments = "-command \"python main.py"";
    GuestProcessManager gpm = gom.getProcessManager(vm);
    long pid = gpm.startProgramInGuest(npa, spec);
    System.out.println("pid: " + pid);

    si.getServerConnection().logout();
  }
}
```

Figure 43 Implementation of scan request handler

Malware Scanner

The malware scanner component is responsible for initiating scan requests to VirusTotal platform to scan the process executable files. It scans both benign and malicious portable executable files. When the malware scan request is initiated the files will be transferred to VirusTotal over http to initiate malware analysis. Python's OS library will be used to get operating system file pattern and provides virtual machine name.

IntelliJ Pycharm IDE is used to develop the python module. The python module checks for the list of portable executables in the C: drive. For each of the file found, path and process name is stored in python list collection. The list of files are then sent to the virus total sandbox analyzer. Below code snippet explains how the list of portable executables is collected.

```
import ScanFile as scan
import os
from fnmatch import fnmatch
import CollectInfo as collect
import InsertData as data
filelist={}
list ={}
root = "C:\\"
pattern = "*.exe"

for path, subdirs, files in os.walk(root):
    for name in files:
        if fnmatch(name, pattern):
            filelist[os.path.join(name)] = path

print filelist
```

Figure 44 Implementation of listing portable executables

VirusTotal Sandbox analyzer

VirusTotal is a free cloud service provided by Google to scan and analyze suspicious files and URLs. The files that are sent to VirusTotal service will be executed in a sandbox environment. VirusTotal aims at providing infrastructure and more reliable platform for Virus detection and malware analysis. VirusTotal exposes set of APIs to interact with. Based on the implementation each API can be called within the limit to scan the files. As per the documentation from VirusTotal, the API call will be made with API key and the file name as the parameters. The code snippet shown below depicts the same.

```
import requests
import simplejson as json
import logging as log
from datetime import datetime
import InsertData as db
import socket
import bson.objectid as objectid
currentScanRequests = {}
# Created by Vinoth Baalaji A S for CMPE 295B Project
def scanfile(filepath, filename):
    json_dbdata={}
    params = {'apikey': '7e869b114cc93514b9dd2c9efc68c863bbaa5a40a1e710b3521b524b5b4bf84f'}
    files = {'file': (filename, open(filepath+"\\"+filename, 'rb'))}
    log.debug("Initiating scan request for file "+filename+"in path"+filepath+"made @ "+ str(datetime.now()))
    response = requests.post('https://www.virustotal.com/vtapi/v2/file/scan', files=files, params=params)
    log.debug("Scan request sent")
    json_response = response.json()
    json_str = json.dumps(json_response)
    json_obj = json.loads(json_str)
    if json_response["response_code"] == 1:
        log.debug("Scan response for file "+filename+"is "+str(json_response))
        scanidval = json_obj["scan_id"]
        scanid = scanidval.split('-', 1)[0]
        currentScanRequests[filename]=scanid.split('-',1)[0]
```

Figure 45 Implementation of virus total api analyzer

RESTful Call: **https://www.virustotal.com/vtapi/v2/file/scan**

Parameters:

params = {**'apikey'**:

'7e869b114cc93514b9dd2c9efc68c863bbaa5a40a1e710b3521b524b5b4bf84f'}

files = {**'file'**: (filename, open(filepath+**"\\"**+filename, **'rb'**))}

response = requests.post(**'https://www.virustotal.com/vtapi/v2/file/scan'**, files=files, params=params)

The JSON response for the above mentioned REST call is mentioned below

```
# Create JSON to insert into MongoDB
        json_dbdata["_id"]=str(objectid.ObjectId())
        json_dbdata["ScanId"]=scanid
        json_dbdata["CreationTime"]=str(datetime.now())
        json_dbdata["APIResponse"]=json_obj
        json_dbdata["FileName"]=filename
        json_dbdata["MachineName"]=socket.gethostname()
        json_dbdata["Status"]="Scan Initiated"
        db.writedata(json_dbdata)
        log.debug("VirusTotal Scan Id is "+scanid)
    else:
        db.writedata(json_obj)
```

Figure 46 Implementation of JSON Response in MongoDB

Behavior Data Extraction Engine

The behavior data extraction engine is responsible for extracting the behavior data from VirusTotal service. Once the scan requests are generated by VirusTotal, the files are executed in a sandboxed environment. The behavior data can be consumed once it is available. The VirusTotal service exposes API to get the behavior report in the form of JSON. The behavior data comprises of network hosts used by the executable file, registry modifications, memory attributes and API call logs used by the executable files. In scope of our implementation, the behavior data extraction engine is designed to extract only the API call logs from the JSON response. The REST call and python code for behavior data extraction from VirusTotal service is mentioned below.

```
def collectbehaviorInfo(id,scanid,filename):
    initiateFeatureVector()
    params = {'apikey': '7e869b114cc93514b9dd2c9efc68c863bbaa5a40a1e710b3521b524b5b4bf84f','hash': scanid}
    response = requests.get('https://www.virustotal.com/vtapi/v2/file/behaviour', params=params)
    json_response = response.json()
    json_str = json.dumps(json_response)
    output=''
    featureOutput=''
    json_obj = json.loads(json_str)
    if "response_code" in json_obj:
        if json_obj["response_code"] == 0:
            data.updateData('scanhistory','No Behavior Data',id)
```

```
else:
    if len(json_obj["behavior"]["processes"]) > 0:
        for each in json_obj["behavior"]["processes"]:
                for  innereach in each["calls"]:
                    if innereach["category"]  == "system":
                        if innereach["status"] != "":
                            val = innereach["arguments"][0]["value"].lower()
                            if val.count("\\") > 0:
                                val = val.split("\\").__getitem__(val.count("\\"))
                            output += val + ','

                            for key,value in feature_hash.items():
                                if(key.lower() == val.lower()):
                                    feature_hash[val]="1"

        for k,v in sorted(feature_hash.items()):
            featureOutput += v+','
        print sorted(feature_hash.items())
        behavior_json["ScanId"]=scanid
        behavior_json["CreationTime"]=str(time.datetime.now())
        behavior_json["MachineName"] = socket.gethostname()
        behavior_json["SysyemDLL"]=output
        behavior_json["CSV"]=filename+','+output
        behavior_json["FeatureVector"]=filename+','+featureOutput
        data.writebehaviordata(behavior_json)
        print filename + ',' + featureOutput
        data.updateData('scanhistory','Analysis in Progress',id)
```

Figure 47 Implementation of data extraction engine

RESTful URL Call: **https://www.virustotal.com/vtapi/v2/file/behaviour**

Parameters:

params = {'apikey':

'7e869b114cc93514b9dd2c9efc68c863bbaa5a40a1e710b3521b524b5b4bf84f','hash': scanid}

Sample JSON output containing behavior data of a process is mentioned below.


```
{u'info':
'behavior': {u'processes': [
{u'parent_id': u'888', u'process_id': u'904', u'process_name': u'8d559882814deb935b902051654c9b95e22dfd0cf303553b8380b1ca3d26f590', u'first_seen':
u'20140802084119.384',u'calls': [{u'category': u'registry', u'status': u'SUCCESS', u'return': u'0x00000010', u'timestamp': u'20140802084119.384', u'repeated':
0, u'api': u'RegOpenKeyExW', u'arguments': [{u'name': u'hKey', u'value': u'HKEY_LOCAL_MACHINE'}, {u'name': u'lpSubKey', u'value': u'SOFTWARE'}]}, {u'category'
u'registry', u'status': u'SUCCESS', u'return': u'0x00000070', u'timestamp': u'20140802084119.384', u'repeated': 0, u'api': u'RegOpenKeyExW', u'arguments':
[{u'name': u'hKey', u'value': u'0x00000010'}, {u'name': u'lpSubKey', u'value':
u'Classes\\CLSID\\{00000315-0000-0000-C000-000000000046}\\Conversion\\Readable'}]}, {u'category': u'filesystem', u'status': u'SUCCESS', u'return':
u'0x00000074', u'timestamp': u'20140802084119.384', u'repeated': 0, u'api': u'CreateFileW', u'arguments': [{u'name': u'lpFileName', u'value':
u'C:\\WINDOWS\\system32\\atl.dll'}, {u'name': u'dwDesiredAccess', u'value': u'0x00000001'}]}, {u'category': u'memory', u'status': u'SUCCESS', u'return':
u'0x00150000', u'timestamp': u'20140802084119.384', u'repeated': 0, u'api': u'VirtualAllocEx', u'arguments': [{u'name': u'th32ProcessID', u'value': u'904'},
{u'name': u'szExeFile', u'value': u'8d559882814deb935b902051654c9b95e22dfd0cf303553b8380b1ca3d26f590'}, {u'name': u'lpAddress', u'value': u'0x00000000'},
{u'name': u'dwSize', u'value': u'255488'}, {u'name': u'flAllocationType', u'value': u'0x00003000'}, {u'name': u'flProtect', u'value': u'0x00000040'}]},
{u'category': u'memory', u'status': u'SUCCESS', u'return': u'0x00190000', u'timestamp': u'20140802084119.394', u'repeated': 0, u'api': u'VirtualAllocEx',
u'arguments': [{u'name': u'th32ProcessID', u'value': u'904'}, {u'name': u'szExeFile', u'value':
u'8d559882814deb935b902051654c9b95e22dfd0cf303553b8380b1ca3d26f590'}, {u'name': u'lpAddress', u'value': u'0x00000000'}, {u'name': u'dwSize', u'value':
u'1024'}, {u'name': u'flAllocationType', u'value': u'0x00003000'}, {u'name': u'flProtect', u'value': u'0x00000040'}]}, {u'category': u'memory', u'status':
u'SUCCESS', u'return': u'0x001a0000', u'timestamp': u'20140802084119.394', u'repeated': 0, u'api': u'VirtualAllocEx', u'arguments': [{u'name':
u'th32ProcessID', u'value': u'904'}, {u'name': u'szExeFile', u'value': u'8d559882814deb935b902051654c9b95e22dfd0cf303553b8380b1ca3d26f590'}, {u'name':
u'lpAddress', u'value': u'0x00000000'}, {u'name': u'dwSize', u'value': u'266240'}, {u'name': u'flAllocationType', u'value': u'0x00003000'}, {u'name':
```

```
u'arguments': [{u'name': u'lpName', u'value': u'Global\\{CB561546-E774-D5EA-8F92-61FCBA8C42EE}'}]}, {u'category': u'synchronization', u'status': u'FAILURE',
u'return': u'', u'timestamp': u'20140802084119.414', u'repeated': 0, u'api': u'OpenMutexW', u'arguments': [{u'name': u'dwDesiredAccess', u'value':
u'SYNCHRONIZE'}, {u'name': u'lpName', u'value': u'Local\\{55E9553D-A70F-4B55-8F92-61FCBA8C42EE}'}]}, {u'category': u'memory', u'status': u'SUCCESS', u'return':
u'', u'timestamp': u'20140802084119.414', u'repeated': 0, u'api': u'ReadProcessMemory', u'arguments': [{u'name': u'th32ProcessID', u'value': u'904'}, {u'name':
u'szExeFile', u'value': u'8d559882814deb935b902051654c9b95e22dfd0cf303553b8380b1ca3d26f590'}, {u'name': u'lpBaseAddress', u'value': u'0x7c811185'}, {u'name':
u'nSize', u'value': u'1'}]}, {u'category': u'synchronization', u'status': u'FAILURE', u'return': u'', u'timestamp': u'20140802084119.414', u'repeated': 0,
u'api': u'OpenMutexW', u'arguments': [{u'name': u'dwDesiredAccess', u'value': u'SYNCHRONIZE'}, {u'name': u'lpName', u'value':
u'Global\\{55E9553D-A70F-4B55-8F92-61FCBA8C42EE}'}]}, {u'category': u'synchronization', u'status': u'FAILURE', u'return': u'', u'timestamp':
u'20140802084119.414', u'repeated': 0, u'api': u'OpenMutexW', u'arguments': [{u'name': u'dwDesiredAccess', u'value': u'SYNCHRONIZE'}, {u'name': u'lpName',
u'value': u'Global\\{55E9553C-A70E-4B55-8F92-61FCBA8C42EE}'}]}, {u'category': u'synchronization', u'status': u'FAILURE', u'return': u'', u'timestamp':
u'20140802084119.414', u'repeated': 0, u'api': u'OpenMutexW', u'arguments': [{u'name': u'dwDesiredAccess', u'value': u'SYNCHRONIZE'}, {u'name': u'lpName',
u'value': u'Local\\{55E9553C-A70E-4B55-8F92-61FCBA8C42EE}'}]}, {u'category': u'synchronization', u'status': u'FAILURE', u'return': u'', u'timestamp':
u'20140802084119.414', u'repeated': 0, u'api': u'OpenMutexW', u'arguments': [{u'name': u'dwDesiredAccess', u'value': u'SYNCHRONIZE'}, {u'name': u'lpName',
```


Figure 48 Sample of data extracted

The following output is the sample of behavior data extracted from the engine.

FileScanned:

{u'578ee1b2ada5b917b4c7352b':

[u'e9b92d775ef6b73b5a28165a367760533e4293c4c4b4a15596a575248b1317d4',

u'essetup.exe']}

Behavior data Output for essetup.exe:

['advapi32.dll', 'cabinet.dll', 'clbcatq.dll', 'comctl32.dll', 'comdlg32.dll', 'crypt32.dll', 'dnsapi.dll', 'hnetcfg.dll', 'kernel32.dll', 'le32.dll', 'midimap.dll', 'msacm.dll', 'msimg.dll', 'mswsock.dll', 'netapi32.dll', 'ntdll.dll', 'oleaut32.dll', 'psapi.dll', 'pstorec.dll', 'rasadhlp.dll', 'riched20.dll', 'rpcrt4.dll', 'rsaenh.dll', 'secur32.dll', 'setupapi.dll', 'shell32.dll', 'shlwapi.dll', 'sxs.dll', 'user32.dll', 'userenv.dll', 'version.dll', 'winhttp.dll', 'wininet.dll', 'winrnr.dll', 'wintrust.dll', 'ws2_32.dll', 'wshtcpip.dll']

Feature Vector Generation Module

The behavior data extracted will consists of all the dlls used by the executable in the sandbox environment. But to identify if the process file is malicious or not, we have to refine these dlls to identify the dlls that are close to malware behavior. A feature vector can be defined as a group of API calls (dlls) that exhibit characteristics similar to any Malware family. The feature vector generation module generates a feature vector for the machine learning algorithm from the behavior data. Following is the sample feature vector generated for a process executable file.

essetup.exe,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,

Data Store and Date Management Module

This module is responsible for maintaining the data generated by the middle tier. All the data that is collected from the above mentioned modules are stored in a NoSQL database. The database is created and maintained in cloud for continuous availability. The following python code snippet is used to write the scan history data to MongoDB

```
def writedata(JSON_Data):
    MONGODB_URI = 'mongodb://mongo:ognom@ds015934.mlab.com:15934/scanreport'
    client = pymongo.MongoClient(MONGODB_URI)
    inpdata = json.dumps(JSON_Data)
    db = client.get_database('scanreport')
    data = db['scanhistory']
    data.insert(json.loads(inpdata))
    client.close()
```

Figure 49 Implementation of data store and management

Query scanId operation:

```
def queryScanIds():
    scaniddictionary={}
    MONGODB_URI = 'mongodb://mongo:ognom@ds015934.mlab.com:15934/scanreport'
    client = pymongo.MongoClient(MONGODB_URI)
    db = client.get_database('scanreport')
    data = db['scanhistory']
    results = data.find({"Status": "Scan Initiated"})
    for result in results:
        scaniddictionary.setdefault(result["_id"], []).append(result["ScanId"])
        scaniddictionary.setdefault(result["_id"], []).append(result["FileName"])
    return scaniddictionary
```

Figure 50 Implementation of queryScanID setting

Insert Behavior analysis data:

```
def writebehaviordata(JSON_Behaviordata):
    MONGODB_URI = 'mongodb://mongo:ognom@ds015934.mlab.com:15934/scanreport'
    client = pymongo.MongoClient(MONGODB_URI)
    inpdata = json.dumps(JSON_Behaviordata)
    db = client.get_database('scanreport')
    data = db['behaviordata']
    data.insert(json.loads(inpdata))
    client.close()
```

Figure 51 Implementation of behavioural data insert.

Update Scan Status:

```
def updateData(Type,Status,Id):
    MONGODB_URI = 'mongodb://mongo:ognom@ds015934.mlab.com:15934/scanreport'
    client = pymongo.MongoClient(MONGODB_URI)
    updateDb=client.get_database('scanreport')
    updateCollection = updateDb[Type]
    updateCollection.update_one({"_id": Id},
    {
        "$set": {
            "Status": Status
        },

    })
```

Figure 52 Implementation of updating scan status

Analytics Tier Implementation

In this section, we explain the detailed description of how data is managed and feed to machine learning algorithm to generate results. The data manipulated as per requirement is feed to machine learning algorithm i.e. WeightClassifier. The generated results from the algorithm are stored into MongoDB.

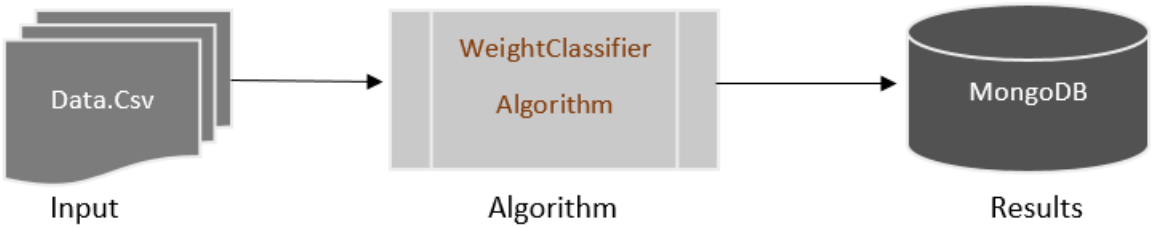


Figure 53 Flow of ML algorithm

Data management

The test and training data are collected from implementation of middle tier (explained in above section). The collected data is in .CSV format with designed feature vector.

The feature vector for our training consist of type of exe files and list of DLLs accessed. The feature vector of test data consists of .exe filenames and the list of DLLs accessed by it. The part of sample training data collected is shown in below figure.

```
train.csv
Type,advapi32.dll,cabinet.dll,clbcatq.dll,comctl32.dll,comdlg32.dll,crypt32.dll,dnsapi.dll,hnetcfg.dll,kernel32.dll,le32.dll,midimap.dll,msacm.d
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,0,1,1,0,0,0,0,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,0
Zbot,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,1,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,1,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
Zbot,1,0,1,1,0,1,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,0,0,0,1
```

Figure 54 Sample of training data

The list of DLLs used in our feature vector are shown in below table.

Advapi32.dll	Cabinet.dll
Clbcatq.dll	Comctl32.dll
Comdlg32.dll	Crypt32.dll
Dnsapi.dll	Hnetcfg.dll
Kernel32.dll	Ole32.dll
Midimap.dll	Msacm.dll
Msimg.dll	Mswsock.dll
Netapi32.dll	Ntdll.dll
Oleaut32.dll	Psapi.dll
Pstorec.dll	Rasadhlp.dll
Riched20.dll	Rpcrt4.dll
Rsaend.dll	Secur32.dll
Setupapi.dll	Shell32.dll
Shlwapi.dll	Sxs.dll
User32.dll	Userenv.dll
Version.dll	Winhttp.dll
Wininet.dll	Winrnr.dll
Wintrust.dll	Ws2_32.dll
Wshtcpip.dll	

Figure 55 List of dlls in feature vector

This list of dll's are total number of dll's accessed by three families of malware Zbot,Winwebsec and Zeroaccess in a perticular format. The three malware families access subset of these dll's in perticular format that is used to predicting it in future through our WeightClassifier machine learning algorithm.

WeightClassifier Algorithm

WeightClassifier is our own classification algorithm to predict the malware exe files in a system. The functioning of this algorithm is based on naive-bayes.The functionality of this algorithm can be divided into two phases, training phase and prediction phase. In training phase, the occurrence probability of each dll's for a perticular family of malware and for total instances is calculated. Later in prediction phase the probabilities of each dlls that are accessed by a perticular exe file are calculated with respect to probability of occurrence of those dll's by each of the malware families. The example below describes the functionality of the algorithm.

Example: Consider there are three families of malwares malware1,malware2 and malware3. The assume there are total six different dll's that are accessed by these malwares in a particular order. The training phase and prediction phase dataset looks as described below.

Training Phase: Consider that there is a .csv training data set. So, the feature vector consists of Labels and six dll's.

Label	dll_1	dll_2	dll_3	dll_4	dll_5	dll_6
Malware1	1	1	0	0	0	0
Malware1	1	1	1	0	0	0
Malware2	0	0	1	1	0	0
Malware2	0	0	1	1	1	0
Malware3	0	0	0	0	1	1
Malware3	0	0	0	1	1	1

Figure 56 Sample of training data set to calculate the weights

In .csv file 1's indicate that perticular malware has accessed those dll's and 0's indicate no access. In these phase of algorithm, it calculates the probability of occurrence of each of the dll's for a perticular malware and for total instances.

Here is the probabilities of occurrence for each dll's in total instances.

$P(dll_1)$	2/6
$P(dll_2)$	2/6
$P(dll_3)$	3/6
$P(dll_4)$	3/6
$P(dll_5)$	3/6
$P(dll_6)$	2/6

Similarly the probabilities of occurrence of each of the dll's for a perticular malware are calculated.

Probability of occurrence of all the dll's by Malware1 (M_1) is

$P(dll_1/M_1)$	2/2
$P(dll_2/M_1)$	2/2
$P(dll_3/M_1)$	1/2
$P(dll_4/M_1)$	0
$P(dll_5/M_1)$	0
$P(dll_6/M_1)$	0

Probability of occurrence of all the dll's by Malware2 (M_2) is

$P(dll_1/M_2)$	0
$P(dll_2/M_2)$	0
$P(dll_3/M_2)$	2/2
$P(dll_4/M_2)$	2/2
$P(dll_5/M_2)$	1/2
$P(dll_6/M_2)$	0

Probability of occurrence of all the dll's by Malware3 (M_3) is

$P(dll_1/M_3)$	0
$P(dll_2/M_3)$	0
$P(dll_3/M_3)$	0
$P(dll_4/M_3)$	1/2
$P(dll_5/M_3)$	2/2
$P(dll_6/M_3)$	2/2

Now, all the calculated weights are stored.

Predicting Phase: In thse phase, the test file with our designed feature vector is considered as input. For all the instances, it calculates the weight of all the dll's accessed by a file and calculate the highest probability. The highest probability is predicted as the one.

Consider the .csv test file with following information. We now need to classify the files into malware categories.

Name	dll_1	dll_2	dll_3	dll_4	dll_5	dll_6
ABC	1	0	0	0	0	0
DEF	0	0	1	0	0	0
GHI	0	0	0	0	1	0

Figure 57 Sample of testing dataset

Now the weights for each of instances to be a particular type of malware is calculated.

Formula to calculate weight

$$\text{Weight}(\text{Instance}_x/M_Y) = \frac{No.of\ instances\ x}{Total\ no\ of\ instances} \times \sum_{i=0}^{6} P(^{dll_i}/_{M_Y})$$

Considered Value of dll_i =1 if it is accessed

=0 if it is not accessed

For instance ABC:

Weight of ABC for M_1 = [(2/6) *((2/2)+0+0+0+0+0)] = 0.33

Weight of ABC for M_2 = [(2/6) *(0+0+0+0+0+0)] = 0

Weight of ABC for M_3 = [(2/6) *(0+0+0+0+0+0)] = 0

Therefore for instance ABC, it has highest weight for Malware1. So, instance ABC is predicted as Malware1.

For instance DEF:

Weight of DEF for M_1 = [(2/6) *(0+0+(1/2)+0+0+0)] = 0.16

Weight of DEF for M_2 = [(2/6) *(0+0+(2/2)+0+0+0)] = 0.33

Weight of DEF for M_3 = [(2/6) *(0+0+0+0+0+0)] = 0

Therefore for instance DEF, it has highest weight for Malware2. So, instance DEF is predicted as Malware2.

For instance GHI:

Weight of GHI for M_1 = [(2/6) *(0+0+0+0+0+0)] = 0

Weight of GHI for M_2 = [(2/6) *(0+0+0+0+(1/2)+0)] = 0.16

Weight of GHI for M_3 = [(2/6) *(0+0+0+0+(2/2)+0)] = 0.33

Therefore for instance GHI, it has highest weight for Malware3. So, instance GHI is predicted as Malware3.

WeightClassifier.Java: The above explained machine learning algorithm is implemented in java language. The implementation of this algorithm can be divided into training phase, testing phase and computing weights phase.

Training Phase:

Here are the data structures, that we have used for implementing WeightClassifier.

'label_instances' is a map, that stores label(String) and the number of corresponding instances(Double)

'label_weights' is a map, where label is mapped to a array of weights. Index of the array corresponds to the index of the feature in the feature vector.

'instances' represents the total number of instances in the train set

'features' indicates the total number of features in the feature vector

```
static HashMap<String,Double> label_instances = new HashMap<>();
static HashMap<String,Double[]> label_weights = new HashMap<>();
static double instances;
static int features;
```

Computing Weights

Training phase is solely relied on computing weights of the train instances. Since 'label_weights' is a map, label(Key of the map) should be unique.

The 'if construct' shown below executes only once for each distinct label, and initializes weight array with feature values. If this label occurs again, the 'else construct' gets executed where in the weights get updated.

The logic discussed above is iterated over all the instances in the train set and 'label_weights' is now a store of label mapped to its weight array.

```
if(!label_weights.containsKey(train_instance[0]))
{
    Double instance_values[] = new Double[features];
    for(int i=0; i<features; i++)
    {
        instance_values[i] = Double.parseDouble(train_instance[i+1]);
    }
    label_weights.put(train_instance[0], instance_values);
    label_instances.put(train_instance[0], 1d);
}
else
{
    Double weights[] = label_weights.get(train_instance[0]);
    for(int i=0; i<weights.length; i++)
    {
        weights[i] = weights[i]+Double.parseDouble(train_instance[i+1]);
    }
    label_weights.replace(train_instance[0],weights);
    Double old_instances = label_instances.get(train_instance[0]);
    label_instances.replace(train_instance[0],old_instances+1d);
}
```

Figure

```
if(!label_weights.containsKey(train_instance[0]))
{
    Double instance_values[] = new Double[features];
    for(int i=0; i<features; i++)
    {
        instance_values[i] = Double.parseDouble(train_instance[i+1]);
    }
    label_weights.put(train_instance[0], instance_values);
    label_instances.put(train_instance[0], 1d);
}
else
{
    Double weights[] = label_weights.get(train_instance[0]);
    for(int i=0; i<weights.length; i++)
    {
        weights[i] = weights[i]+Double.parseDouble(train_instance[i+1]);
    }
    label_weights.replace(train_instance[0],weights);
    Double old_instances = label_instances.get(train_instance[0]);
    label_instances.replace(train_instance[0],old_instances+1d);
}
```

Figure 58 Implementation for computing weights

Testing phase

'computeScores(indexes)' is the entry point for each test instance, where in 'indexes' specifies the collection of feature indexes(only features that exist in this test instance).

Score for each label is computed based on 'label_weights' along with 'indexes'

'label_score' is the data structure for storing the labels and their respective scores. After collecting the scores for each label, 'label_score' is passed to 'computeHighestScore(label_score)' which returns the label with the highest score. This label is the predicted label for test instance.

```
public static String computeScores(ArrayList<Integer> indexes)
{
    HashMap<String,Double> label_score = new HashMap<>();
    Set<String> labels = label_weights.keySet();
    for(String label: labels)
    {
        Double labelscore = label_instances.get(label)/instances;
        Double weights[] = label_weights.get(label);
        for(Integer index: indexes)
        {
            labelscore = labelscore*(weights[index]/label_instances.get(label));
        }
        //System.out.println(label+" : "+labelscore);
        label_score.put(label, labelscore);
    }
    return computeHighestScore(label_score);
}

public static String computeHighestScore(HashMap<String, Double> label_score)
{
    Set<String> labels = label_score.keySet();
    Iterator labelsIterator = labels.iterator();
    String highestScoredLabel = (String)labelsIterator.next();

    while(labelsIterator.hasNext())
    {
        String tempLabel = (String)labelsIterator.next();
        if(label_score.get(tempLabel) > label_score.get(highestScoredLabel))
        {
            highestScoredLabel = tempLabel;
        }
    }

    return highestScoredLabel;
}
```

Figure 59 Implementation for Testing

Results into MongoDB

The generated result file from machine learning algorithm are stored in perticular location in a virtual machine. The results are next stored into MongoDB for future purpose to dispay in UI. The input for this java code is the results genrated earlier. The sample of input is shown below

Filename	Label
spin3d.exe	Benign
kbsetup.exe	Benign
easetup.exe	Benign
frostwire.exe	Winwebsec
7b0c7f6c5b4c392377fb8a1cb9866a23b736a208.exe	Zeroaccess
797ac6d6daa9c3964f3cb20354110c3a95c48a5d.exe	Zeroaccess
7c1413f35187b67690ee9775ba7a66f051353422.exe	Zeroaccess
7ba87c340f44d31acb3b03e8f6016335f7119a8b.exe	Zeroaccess
5fbbcdb7f9a6ceed615e4710f7090be7d77076b4.exe	Winwebsec
608c35c414fb682984745ffdec44f6dff1d723bd.exe	Winwebsec
616cfe9920e75d5f6f5f23ba9182535128bd84d9.exe	Winwebsec
6074237ed5a21fd64c2aaa9cab27803569457ac9.exe	Winwebsec
47df6ae36907d3cbed811cf380dd7e4a808f02f8.exe	Zeroaccess
4ab308c2cf44b6400e022597143f962dec74b87a.exe	Zbot
482dec8ceabb7461e97e5d86f071d002cab6dc1d.exe	Zbot
4918649e00da4be6e5b335dd815a202705e1a4dd.exe	Zbot

Figure 60 screenshot of Testing Results

These data in result.csv file is stored in MongoDB using java script explained below.

Connecting to MongoDB database and accessing the “mycol” collection.

```
try{

    // To connect to mongodb server
    MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

    // Now connect to your databases
    MongoDatabase db = mongoClient.getDatabase( "test" );
    System.out.println("Connect to database successfully");

   // boolean auth = db.authenticate(myUserName, myPassword);
    //System.out.println("Authentication: "+auth);

   //db.createCollection("mycol");
    MongoCollection<Document> coll = db.getCollection("mycol");
    //System.out.println("Collection created successfully");
```

Pushing filenames and labels into MongoDB.

```
MongoCollection<Document> coll = db.getCollection("newColl");
//System.out.println(coll.g);
Date sysdate = new Date();
Document doc = new Document();
doc.put("Filename", document[0]);
doc.put("CreationDate", sysdate.toString());
doc.put("Label", document[1]);
//doc.put("third col",documet[2]);
System.out.println("inserting " + document[0] + " " + document[1]);
coll.insertOne(doc);
```

Figure 61 Implementation for pushing results to mongo

Results stored into MongoDB.

Figure 62 Final results in mongo

MongoDB

MongoDB is NoSQL database, which supports high throughput and increase database performance. It's an open source platform document oriented database that favors JSON like data in form of documents. Each instance of MongoDB consists of databases, collections and documents. These terminologies are analogous to databases, tables and records in relational terminology. In our implementation, MongoDB is used to store the JSON like behavior data sent by the VirusTotal service. As the data is non-relational, the implementation is made using MongoDB. Following are some of the characteristics of MongoDB that were convincing for our implementation

- Behavioral data is non-relational, which forced us to implement a document oriented database.
- The data has to be available seamlessly for the machine learning algorithm to consume the data for analysis.
- The volume could increase overtime and hence a load balanced and scalable data store is necessary.

Java Script Object Notation is a message exchange format used today. It makes it easy for browser to server to exchange data and primarily used in RESTful API calls. The document in MongoDB will be written in JSON format. The database setup is shown as below

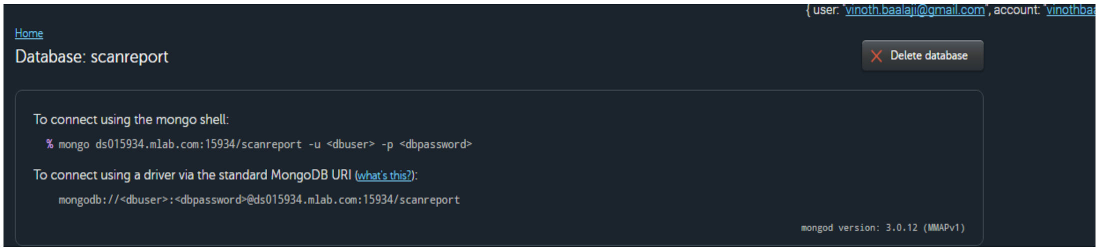


Figure 63 Snapshot of the mongoDB database

Collections:

Following are the collections used in MongoDB database. The data stored in each collection is categorized as below.

- Behaviordata – Stores JSON data of behavior information collected from VirusTotal
- Results – Malware analysis results from Machine Learning algorithm,
- Scanhistory – JSON documents with list of all scan ids of the process executables.

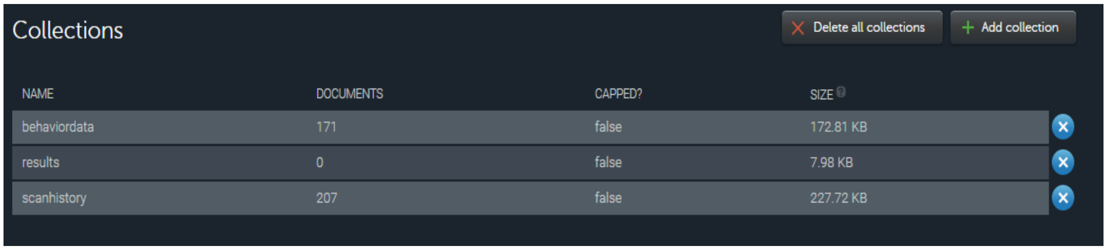


Figure 64 List of various mongo DB collections used

JSON Structures

Behavior data:


```
{
    "_id": {
        "$oid": "578ee7bdada5b92808f6ebdf"
    },
    "ScanId": "3928b4807f525649d9c6ba9499655f9ee13f23cab05d8c1d8f9cf406266680db",
    "SysyemDLL": "shell32.dll,comctl32.dll,advapi32.dll,imm32.dll,imm32.dll,uxtheme.dll,shell32.dll,shfolder.dll,kernel32.dll,uxtheme.dll,",
    "MachineName": "Vinu",
    "CreationTime": "2016-07-19 19:53:48.102000",
    "FeatureVector": "Acrylic.exe,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,",
    "CSV": "Acrylic.exe,shell32.dll,comctl32.dll,advapi32.dll,imm32.dll,imm32.dll,uxtheme.dll,shell32.dll,shfolder.dll,kernel32.dll,uxtheme.dll,"
}
```

Figure 65 Sample of behavioral data stored in Mongo DB

Scan History:


```
{
    "_id": "578ee7b8ada5b92808f6ebde",
    "Status": "No Behavior Data",
    "ScanId": "29f981c59e6979290c47488f6120950f76083ac8fbb6ac544b66d8dee871bd9f",
    "MachineName": "Vinu",
    "CreationTime": "2016-07-19 19:53:44.968000",
    "APIResponse": {
        "permalink": "https://www.virustotal.com/file/29f981c59e6979290c47488f6120950f76083ac8fbb6ac544b66d8dee871bd9f/analysis/1468983156/",
        "sha1": "d7b209d32b49ff75a28b9e76debf78e7072ac2f5",
        "resource": "29f981c59e6979290c47488f6120950f76083ac8fbb6ac544b66d8dee871bd9f",
        "response_code": 1,
        "scan_id": "29f981c59e6979290c47488f6120950f76083ac8fbb6ac544b66d8dee871bd9f-1468983156",
        "verbose_msg": "Scan request successfully queued, come back later for the report",
        "sha256": "29f981c59e6979290c47488f6120950f76083ac8fbb6ac544b66d8dee871bd9f",
        "md5": "ca4285f3144419a25d92e51b713c4061"
    },
    "FileName": "abdioReader.exe"
}
```


Figure 66 Sample of Scan History Details

Results:

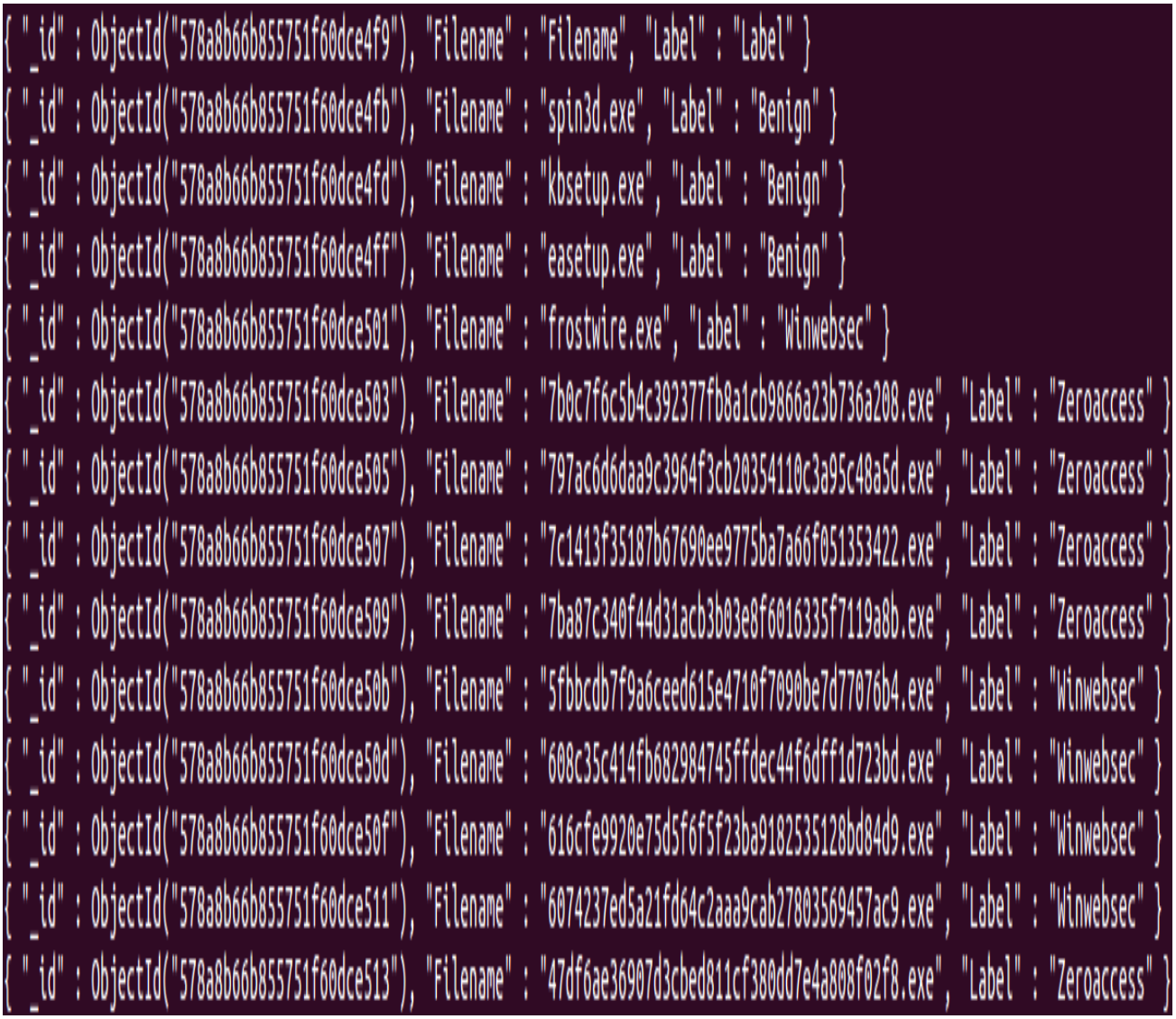


Figure 67 Sample of Machine learning algorithm result

Chapter 6 Testing and Verification

Test Case for Client Tier

Test Case ID: 1	Name: Login
Purpose	When admin want to use the application he can login using his credentials
Pre-requisite	Application must be running state, Mongo DB must be up.
Priority	Medium
Steps	1. Login using admin's credentials 2. Authorize user 3. If new user, register user as Admin.
Result	Admin details are stored in database
Tester Name	Pooja Takavale

Test Case ID: 2	Name: VMs Details on Screen
Purpose	When the admin logins to the application, he must be able to see the VMs in the network
Pre-requisite	VCenter is up and Connected, Application is unning
Priority	High
Steps	1. Login using credentials 2. Authorize user 3. If connection is there, display the VM details on page.
Result	VM Details are etched from the VCenter
Tester Name	Pooja Takavale

Test Case ID: 3	Name: Graphs Statistics
Purpose	When a user logind to the system, he should see the latest graphs as per the VCenter values.
Pre-requisite	Vcenter is up, application is running, Values are fetched at latest time interval
Priority	Medium
Steps	1. Login using credentials

	2. Authorize user 3. VCenter connection is up and user can fetch the latest values to populate the graph using those values. 4. List the graphs.
Result	Graphs metrics are populated using the Highcharts.
Tester Name	Pooja Takavale

Test Case ID: 4	**Name: Scan Results**
Purpose	When a user want to see the scan results of the Machine scanned.
Pre-requisite	Application is running, VCenter is connected and Scanning is performed successfully.
Priority	Medium
Steps	1. Login using google credentials 2. Authorize user 3. When the VM is scanned, the scan results are populated.
Result	Scan results are stored in MongoDB
Tester Name	Pooja Takavale.

Test cases for Middle Tier

Test Case ID:	1
Test Case Name:	Check the list of files picked by process watcher
Purpose:	The malware scanner looks for list of process executable files in the virtual machine. It should pick only .exe files for scanning.
Pre-requisite	Scan request has been already made and the virtual machine is up and running
Priority	Medium
Steps	1. Login to web application 2. select the virtual machine to scan 3. Click on Scan button 4. Check malware scan history in the web application
Result	Malware scanner should have picked only .exe files for scanning

Test Case ID:	2
Test Case Name:	Accessing VirusTotal service with API Key
Purpose:	The malware scanner service is unique for every user. It is

	important to test the authenticity of the service to ensure that the data is not manipulated during exchange
Pre-requisite	File is identified for scan
Priority	High
Steps	1. User clicks on the scan button from UI 2. Malware scanner initiates a scan request to VirusTotal service. 3. The correct API Key is provided in the RESTful call.
Result	Input: Correct API Key Output: Scan Request is authenticated for scan Input 2: Incorrect API Key Output 2: Scan Request is rejected and by VirusTotal API

Test Case ID:	3
Test Case Name:	Scan request generation
Purpose:	For each of the process executable files identified in a VM, scan requests should be generated by VirusTotal API
Pre-requisite	Request is authenticated with right API key
Priority	High
Steps	1. 1. User clicks on the scan button from UI 2. Malware scanner initiates a scan request to VirusTotal service with the right API key 3. File is transferred to VirusTotal API for execution
Result	VirusTotal service generated a unique scan id and sends JSON response to Malware scanner module. The JSON output is then stored in Mongo database.

Test Case ID:	4
Test Case Name:	Query list of scan ids
Purpose:	To validate if all scan ids from Mongo database are valid in order to get the behavior report.
Pre-requisite	Scan Id is generated by VirusTotal API
Priority	High
Steps	1. Login to web application 2. Select a VM 3. Click on the scan history tab
Result	List of all scan ids for that particular VM should be displayed

Test Case ID:	5
Test Case Name:	Behavior report generation for response code 1
Purpose:	To validate and handle all exceptions from VirusTotal JSON response

Pre-requisite	Scan request is initiated and scan id is provided by VirusTotal
Priority	High
Steps	1. Scan Id is generated and stored in MongoDb 2. For one of the process executable file, validate the response coming from VirusTotal 3. Response should be a valid JSON output
Result	Response code value in the JSON output should be 1, which ensures that the behavior data is available.

Test Case ID:	6
Test Case Name:	Validate report for response code 0
Purpose:	To check the behavior data for JSON response with code 0
Pre-requisite	Scan request is initiated and scan id is provided by VirusTotal
Priority	High
Steps	1. Scan Id is generated and stored in MongoDb 2. For one of the process executable file, validate the response coming from VirusTotal 3. Response should be a valid JSON output
Result	If the response code is 0, it implies that there is no behavior report for that process executable file. No behavior data should be generated and the scan history was updated to “No behavior data’ for that particular scan id

Test Case ID:	7
Test Case Name:	No behavior data with response code 1
Purpose:	To handle program flow if the response code is 1 but behavior data is missing in the JSON response
Pre-requisite	Scan request is initiated and scan id is provided by VirusTotal
Priority	High
Steps	1. Scan Id is generated and stored in MongoDb 2. For one of the process executable file, validate the response coming from VirusTotal 3. Response should be a valid JSON output
Result	The response code is 1, JSON output was validated. In this case, the status in scan history will still be updated to No Behavior data.

Test Case ID:	8
Test Case Name:	Feature vector output generation
Purpose:	To validate the feature vector generation based on the API calls present in the JSON response.
Pre-requisite	The VirusTotal service response is obtained by the Malware scanner.

Priority	High
Steps	1. For each process executable files, check the behavior data. 2. Malware scanner extracts all the API call logs from the JSON response.
Result	Feature vector is generated with filename followed by binary values denoting the presence or absence of dll for that process file

Test cases for Data Tier

Test Case ID:	9
Test Case Name:	Machine Learning
Purpose:	Run machine learning algorithm with no train data
Pre-requisite	Install java in virtual machine
Priority	High
Steps	Run Weightclassifier algorithm
Result	No train data.

Test Case ID:	10
Test Case Name:	Machine Learning
Purpose:	Run machine learning algorithm with no test data
Pre-requisite	Install java in virtual machine
Priority	High
Steps	Run Weightclassifier algorithm
Result	No test data.

Test Case ID:	11
Test Case Name:	Machine Learning
Purpose:	Run machine learning algorithm with no header in train data
Pre-requisite	Install java in virtual machine
Priority	High
Steps	Run Weightclassifier algorithm
Result	Improper train data.

Chapter 7 Performance and Benchmarks

Various performance evaluation metrics and benchmarks which we have considered for our project is explained briefly in this section. Our benchmarks was to evaluate the machine learning algorithm for malware detection and overall performance of our detection system.

Experiments & Results

During our study on Malwares, we wanted to identify patterns in their behavior. The following aspects of a malware program execution were considered. They include File system, process monitor, network connection monitor, registry level changes and performance changes during malware program execution.

As an experiment, we simulated benign and malware process in a sandbox environment. We took the file system logs, registry changes and performance values after execution. The values were identical for both benign and malicious executables. Thus, we had to look at other program execution aspects to differentiate a malware from benign process. Further readings and references from various sources we understood that API call logs is an important aspect for malware analysis. When we verified the API call logs from malware simulation, we could find that each malware family invoked a set of dlls.

Metrics for Performance Evaluation

Web Application

It allows checking the Web Application for Load Testing, Stress Testing and Latency time information. We used Eclipse to generate SOAP Request and then we fed that request into JMeter to watch out Latency for multiple requests for a given method. We also performed Stress /Load testing by running 100s of threads accessing web service simultaneously and checking threshold and performance of our system.

Confusion matrix

		PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

$$Accuracy = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

TOTAL NO OF TRAINING SAMPLES	NO.OF. TEST SAMPLES	NO OF TRUE-POSITIVES (TP)	NO OF FALSE POSITIVES (FP)	NO OF TRUE NEGATIVES (TN)	NO OF TRUE-NEGATIVE (FN)	TOTAL POSITIVES (OP)	TOTAL NEGATIVES (ON)	ACCURACY= (TP+TN)/TOTAL	TPR=TP/OP	TNR=TN/ON
100	20	15	1	3	1	16	4	0.9	0.9375	0.75
250	35	21	2	11	1	23	12	0.914	0.91304348	0.916666667
500	50	33	3	12	2	36	14	0.9	0.91666667	0.857142857

Figure 68 Sample of Machine learning algorithm result

The accuracy of the system can be measured by the true positive rate to true negative rate in the classifications made by the algorithm.

Limitation of accuracy:

Consider a class 0 with: 9990 records and class 1 with: 10 record model will predict everything to be class 0 and accuracy can be 99.9%

Cost sensitive measures

Measuring precision and recall of the algorithm can be one method of cost sensitive measures.

$$Precision(p) = \frac{TP}{TP + FP}$$

$$Recall(r) = \frac{TP}{TP + FN}$$

ROC curve

ROC curve is a graphical plot that illustrates the performance of the binary classifier system as its discrimination threshold is varied. Plotting the TPR against the false FPR at various threshold settings creates the curve.

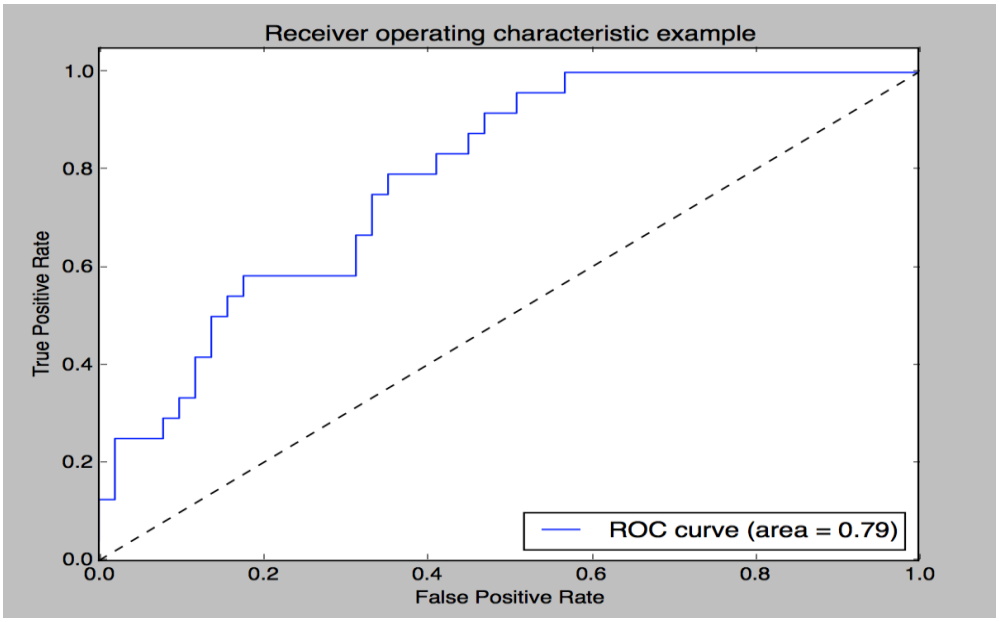


Figure 69 ROC curve results

Benchmarks

Some of the benchmarks that we have to keep in mind for effective evaluation of our project successes are:

- Accuracy of the classification model: The approximate accuracy of the classification model should be in high 90% so establishing it as a successful model.
- The runtime taken for malware detection: The time taken by the algorithm to detect a malicious software running in a particular VM is very important so the effectiveness of our malware detection system.
- Number of Test data: The classification algorithm should be properly tested with high number of test data. Running as many possible samples of malwares that we can collect can improve the accuracy of our classification system.

The chart below elucidates the false positivity of our algorithm and accuracy benchmark of our algorithm.

NO OF ZBOT SAMPLES	NO OF WINBEBSEC SAMPLES	NO OF ZEROACCESS SAMPLES	BENIGN SAMPLES	TOTAL NO OF TRAINING SAMPLES	NO.OF. TEST SAMPLES	NO OF TRUE-POSITIVES	NO OF FALSE POSITIVES	ACCURACY ACHIEVED
25	25	25	25	100	20	14	6	75%
50	75	75	50	250	35	28	7	80%
125	125	125	125	500	50	45	5	90%

For each iteration of our algorithm, we considered different number of samples for analysis. As usual machine learning algorithm practices, the ratio of test and train data is 20:80. So, we trained our algorithm with 100 different samples of malwares. In order to test the malware results, we loaded up to 20 samples of test data consisting of both benign and malware samples. The results provided up to 14 true positives and 6 false positive data.

In the second iteration of our malware scan, we increased the number of training data samples to 250. Feature vector for each of the 250 malware and benign samples were loaded to

machine learning algorithm and our accuracy improved to approximately 80%. To obtain better results, we increased our training sample to up to 500 samples and we were able to obtain 90% accuracy in detecting malicious software.

Chapter 8 Deployment, Operations, Maintenance

The infrastructure and Virtual machine setup for our project is done on VMWare vCenter. Though we have other alternatives like AWS and OpenStack, we chose vCenter because of the fact that we will be allowed to install Malwares and associated programs with ease. VSphere client is an additional software tool that provides the access to vCenter to manage virtual machines. Details such as vCenter IP and number of hosts under vCenter are described below.

Infrastructure Deployment

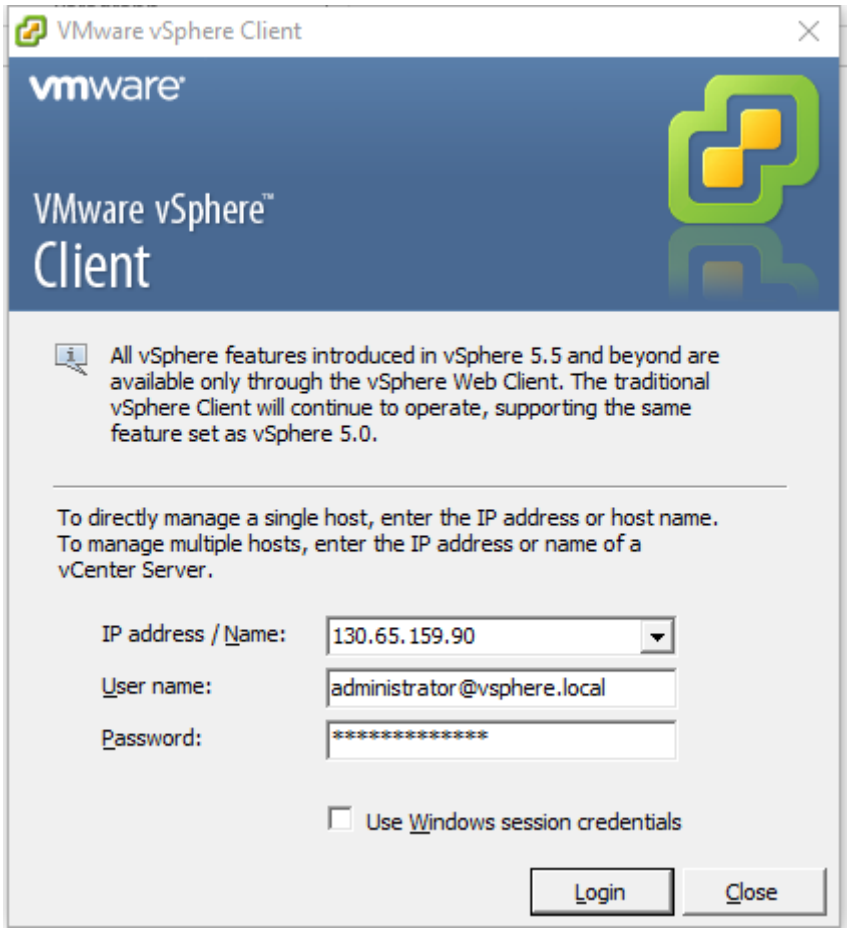


Figure 70 Screenshot of Vsphere login

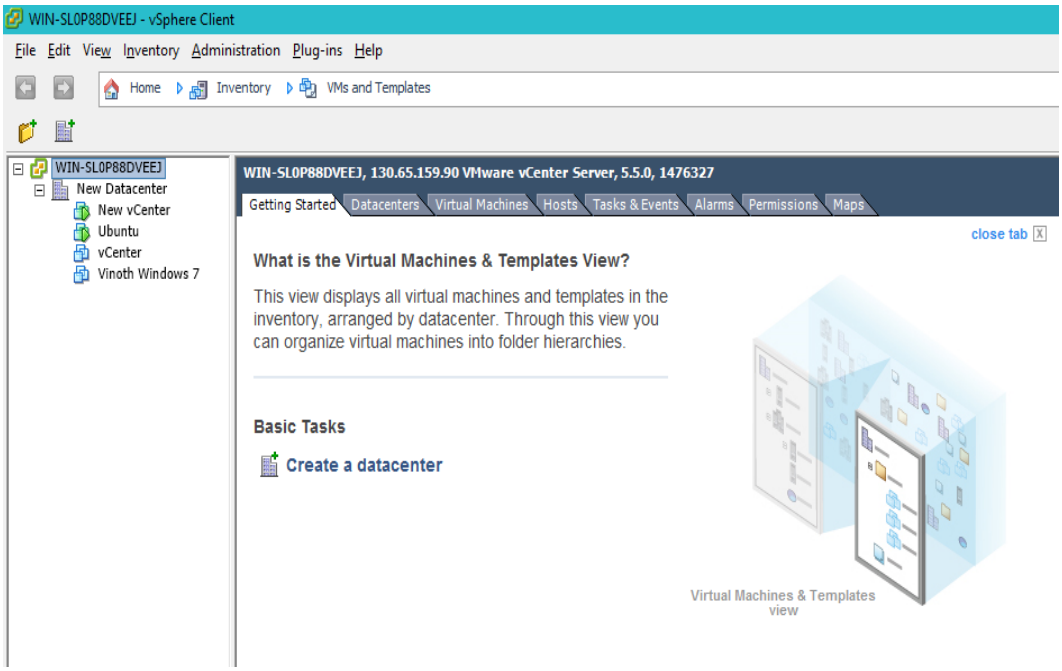


Figure 71 Screenshot of Vsphere client dashboard

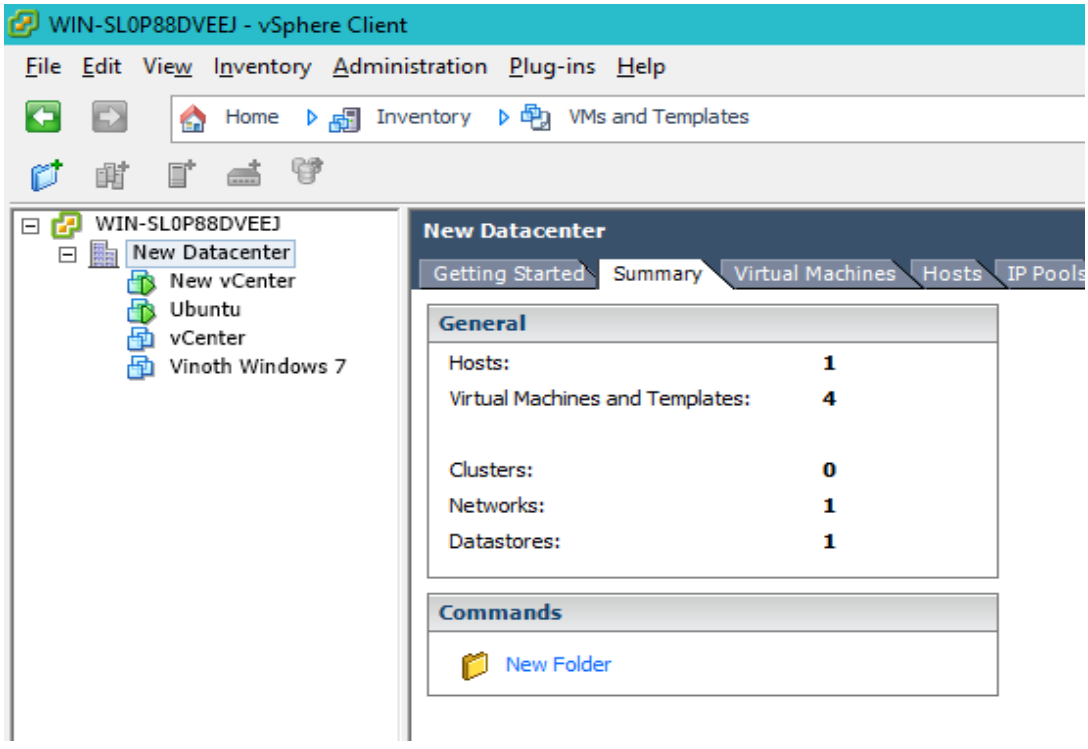


Figure 72 Screenshot of Datacenter details

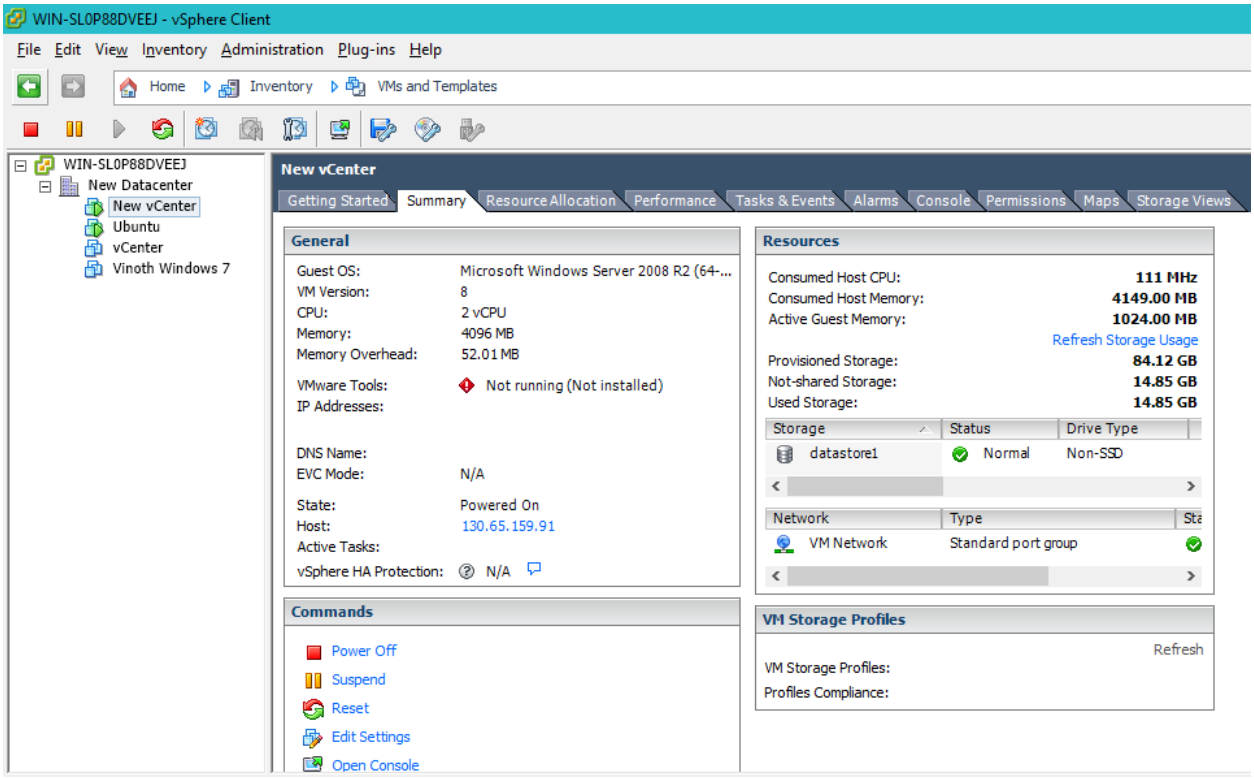


Figure 73 Screenshot of VM1 details

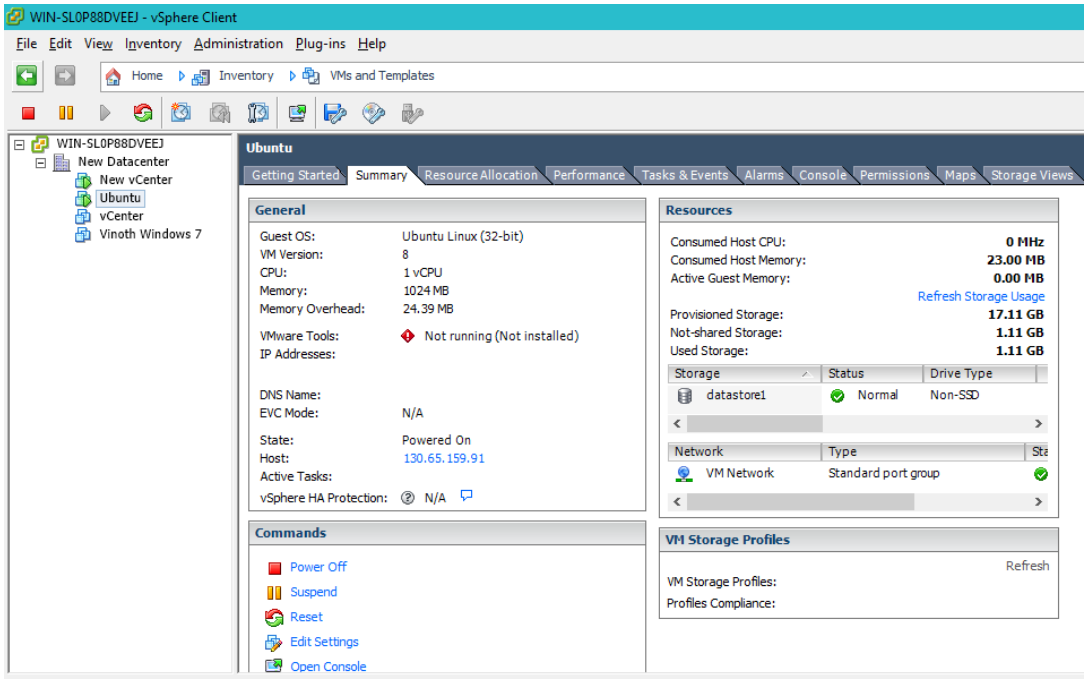


Figure 74 Screenshot of VM2 details

Web Application Deployment - AWS EC2

The dashboard based web application will be deployed on AWS EC2 outside vCenter. This is to alleviate any impacts that could be caused by Malwares. The web application acts as a console for administrators to monitor virtual machines and manage basic VM functions. The detailed deployment steps of creating a EC2 instance is mentioned below.

Added a user (IAM) to the group with admin rights for managing various instances in the aws account.

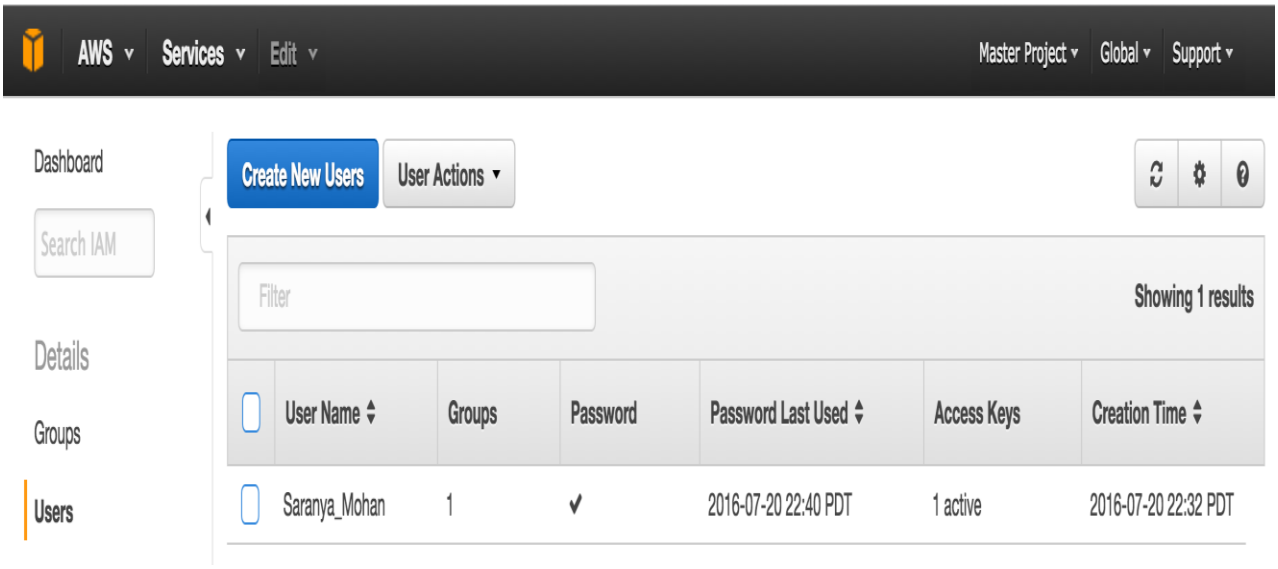


Figure 75 Created a user in group in AWS

Created a Key-Pair for our instances.

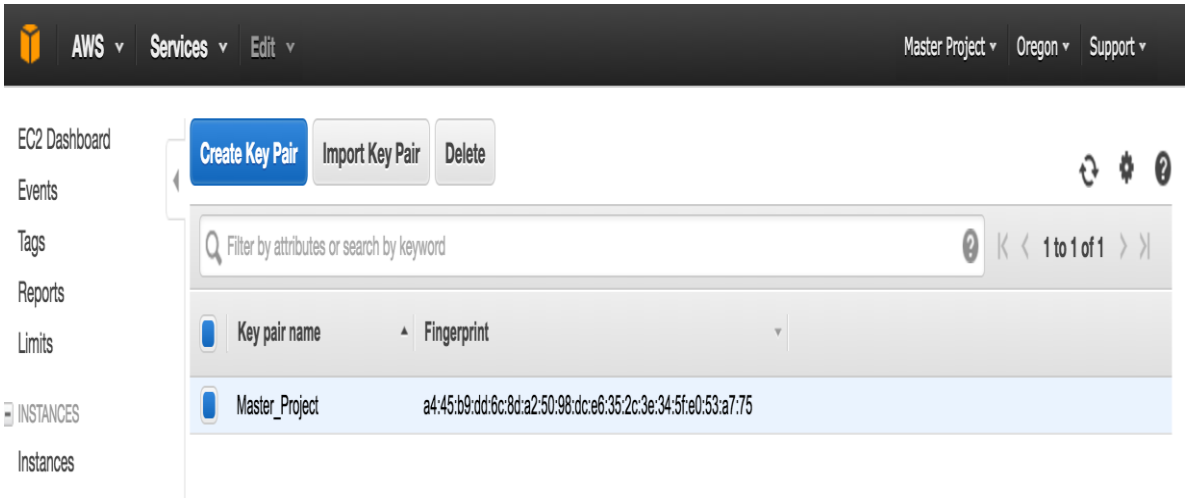


Figure 76 Created a Key-Pair for our instances

Create a security group for the instance which allows access through http, https and ssh protocols .

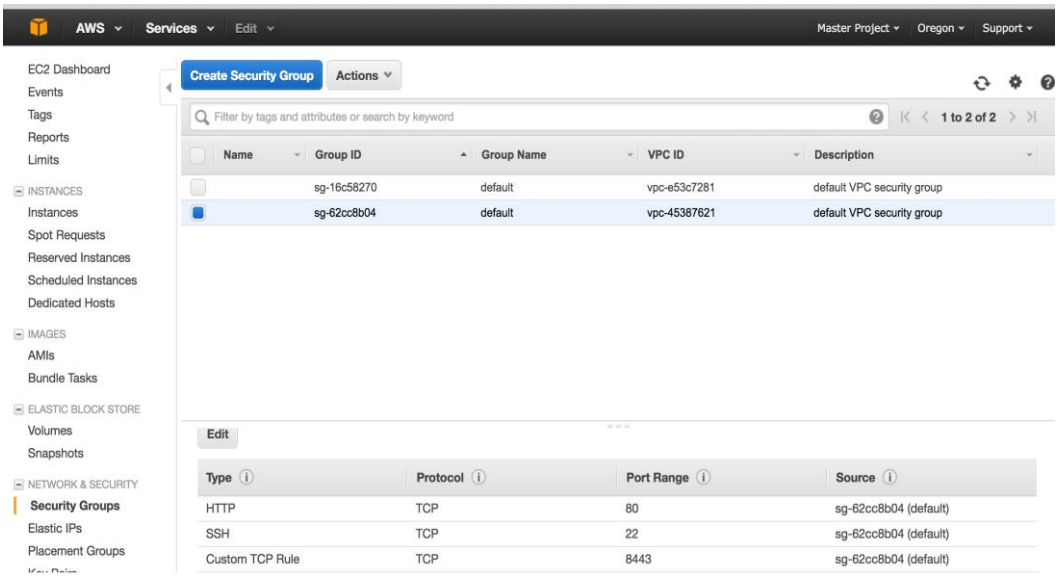


Figure 77 Created a Security Group for our instances

Created a VPC (Virtual private Cloud) for our instance and selecting the already created security group for our instance.

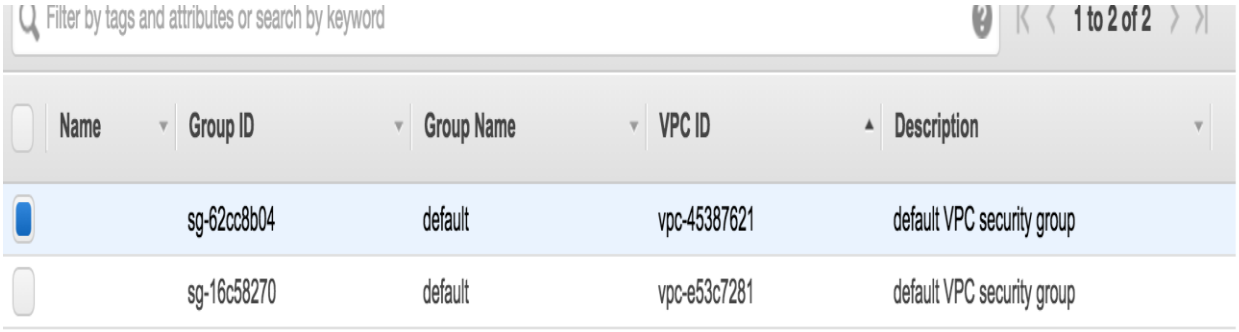


Figure 78 Created a VPC for our instances

Created the EC2 instance for launching the instance

An EC2 instances was successfully launched and is up and running for launching our web application.

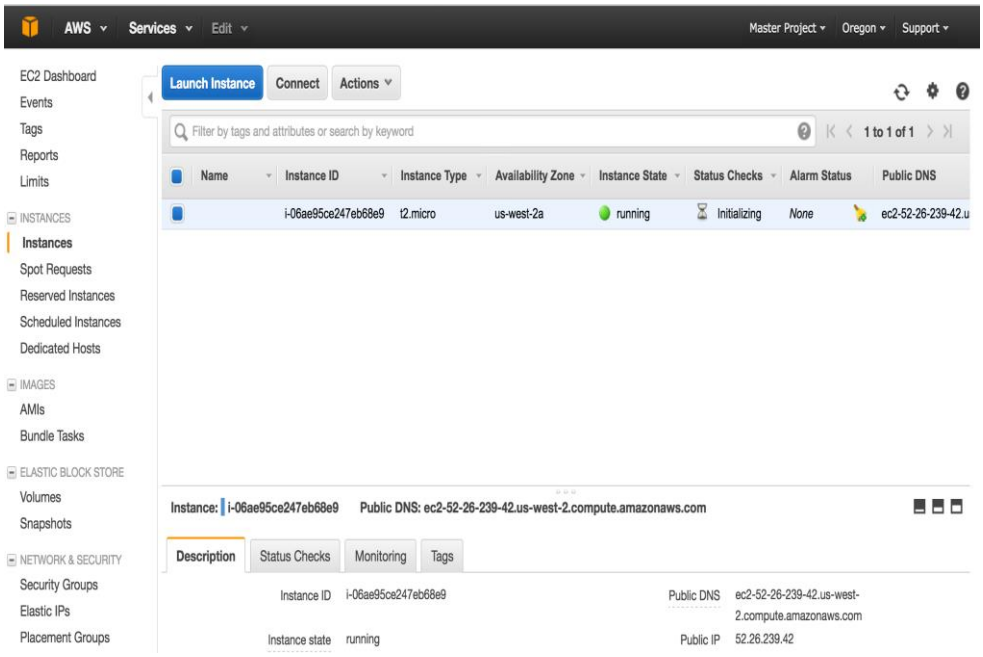


Figure 79 Created a instances which is running successfully

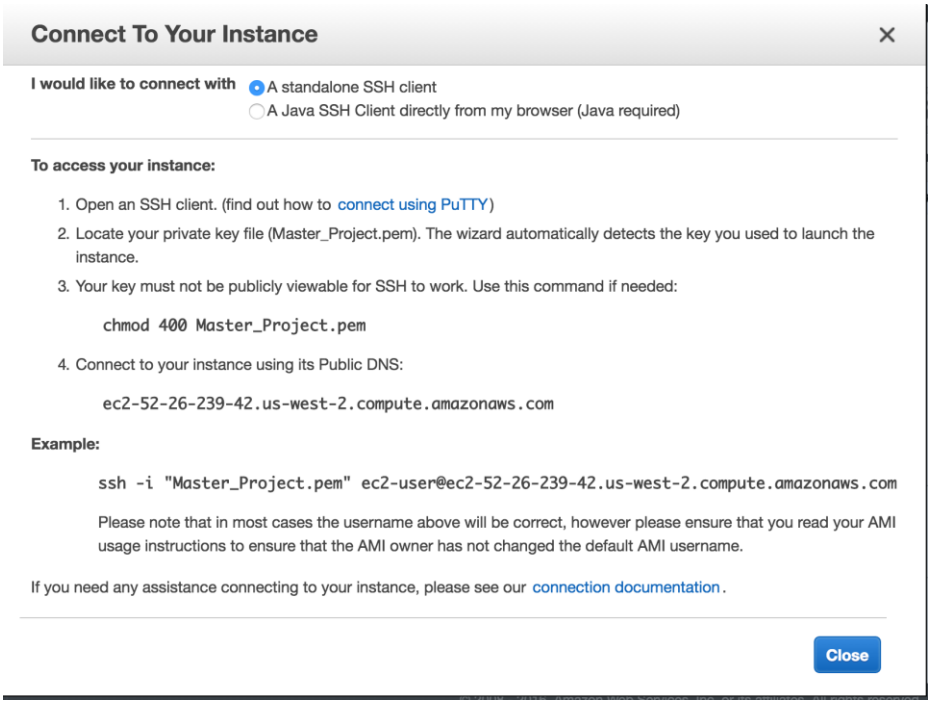


Figure 80 Created a instances which is running successfully

Chapter 9 Summary, Conclusions, and Recommendations

Summary

With increasing number of malwares and their polymorphic behaviors, a powerful detection algorithm is required. Signature based detection is a rigid and hardcoded way of detecting malwares. A dynamic and behavioral malware detection approach provides an innovative way to detect malwares by identifying patterns. From our implementation of malware analysis using machine learning algorithms, we could infer that these algorithms are powerful and more accurate way of detecting malwares on the fly. Our malware detection system is capable of identifying malwares from different families. The machine learning algorithm was able to accurately identify the benign and malicious malwares. Our training data for the machine learning algorithm consists of a subset of malware from variety of malware families. With our initial test data for the machine learning algorithm, the malware detection accuracy was around 70%. In order to improve the accuracy, we increased the malware data samples to nearly 400. This improved the accuracy to 90%. Thus, more the sample dataset more accurate is the malware detection. Finally, we summarize by saying that machine learning algorithm can be considered a better alternative for traditional malware detection mechanisms.

Conclusion

In the context of cloud computing, where a number of virtual machines share the same infrastructure, an intelligent and sophisticated way of detecting malwares is necessary. Machine learning is the way to go due to the latest advancements and capabilities of computing. With this implementation it is evident that our approach can alleviate problems of traditional signature based detection and ensure higher accuracy and intelligent way of detecting malwares.

Recommendations of Future Research

Our current implementation uses API call logs to detect malwares in a Virtual Machine. However, in order to improve the accuracy and improved results, it is important to consider other aspects of an operating system such as network data, VM performance statistics and registry changes made by an executable process. Also, with more powerful machine learning algorithms, our approach can be extended to analyze more malicious families and secure the virtual machines in a better way. Finally, machine learning algorithms can be improved to not only detect the existing malwares but it can also identify any undefined malwares to achieve zero-day malware detection.

Glossary

Hypervisor – A software similar to Operating system, that effectively manages process, resources and applications running multiple virtual machines. E.g. VMWare, Xen, Hyper-V

Virtual Machine – An abstract machine, that runs on a Virtual Machine Monitor (VMM) and shared hardware components. Multiple abstract machines share the same hardware components by virtually running as isolated instances.

AngularJS – A JavaScript framework developed and contributed by Google to improve performance client-server interaction in web applications.

Bootstrap – A front-end development web framework to develop applications quickly and easily.

MongoDB – An open source NoSQL and document driven database, which can replicate the data to handle large number of read requests.

KDD – Refers to the process of finding valuable information from data using data mining techniques.

References

[1] M. Watson, N. u. h. Shirazi, A. Marnerides, A. Mauthe and D. Hutchison. Malware detection in cloud computing infrastructures. *IEEE Transactions on Dependable and Secure Computing PP(99),* pp. 1-1. 2015, .DOI: 10.1109/TDSC.2015.2457918.

[2] Xiaoguang Han, Jigang Sun, Wu Qu and Xuanxia Yao. Distributed malware detection based on binary file features in cloud computing environment. Presented at Control and Decision Conference (2014 CCDC), the 26th Chinese. 2014, DOI: 10.1109/CCDC.2014.6852896.

[3] Jonghoon Kwon and Heejo Lee. BinGraph: Discovering mutant malware using hierarchical semantic signatures. Presented at Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on. 2012, DOI: 10.1109/MALWARE.2012.6461015.

[4] S. Kumar, C. Rama Krishna, N. Aggarwal, R. Sehgal and S. Chamotra. Malicious data classification using structural information and behavioral specifications in executables. Presented at Engineering and Computational Sciences (RAECS), 2014 Recent Advances in. 2014, DOI: 10.1109/RAECS.2014.6799525.

[5] A. J. Duncan, S. Creese and M. Goldsmith. Insider attacks in cloud computing. Presented at Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on. 2012, .DOI: 10.1109/TrustCom.2012.188.

[6] S. Burji, K. J. Liszka and C. C. Chan. Malware analysis using reverse engineering and data mining tools. Presented at System Science and Engineering (ICSSE), 2010 International Conference on. 2010, . DOI: 10.1109/ICSSE.2010.5551719.

[7] Qingshan Jiang, Nancheng Liu and Wei Zhang. A feature representation method of social graph for malware detection. Presented at Intelligent Systems (GCIS), 2013 Fourth Global Congress on. 2013, . DOI: 10.1109/GCIS.2013.28.

[8] D. Komashinskiy and I. Kotenko. Malware detection by data mining techniques based on positionally dependent features. Presented at Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on. 2010, . DOI: 10.1109/PDP.2010.30.

[9] M. G. Schultz, E. Eskin, F. Zadok and S. J. Stolfo. Data mining methods for detection of new malicious executables. Presented at Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. 2001, DOI: 10.1109/SECPRI.2001.924286.

[10] P. M. Comar, L. Liu, S. Saha, P. N. Tan and A. Nucci. Combining supervised and unsupervised learning for zero-day malware detection. Presented at INFOCOM, 2013 Proceedings IEEE. 2013, . DOI: 10.1109/INFCOM.2013.6567003.

[11]K. Takeda and M. Mizutani. Design and prototyping of framework for automated continuous malware collection and analysis. Presented at Security Technology (ICCST), 2011 IEEE International Carnahan Conference on. 2011, . DOI: 10.1109/CCST.2011.6095922.

[12] Cloud-Computing: An industry In Exponential Growth | Investopedia. (2015, March 27).from http://www.investopedia.com/articles/investing/032715/cloudcomputing-industry-exponential-growth.asp

[13] 3 Reasons Why Detecting Cloud Malware Is Essential for Cloud Security Solutions | CipherCloud. (2014, October 27). Retrieved from http://www.ciphercloud.com/blog/3-reasons-why-detecting-cloud-malware-is-essential-for-cloud-security-solutions

[14] Malware attacks businesses as they sleep and cloud remains a threat, researchers say | ZDNet. (2014, July 10). Retrieved from http://www.zdnet.com/article/malware-attacks-businesses-as-they-sleep-and-cloud-remains-a-threat-researchers-say

[15] Trojan.Zeroaccess. (2013, November 29). Retrieved May 12, 2016, from https://www.symantec.com/security_response/writeup.jsp?docid=2011-071314-0410-99