DOI: 10.21917/ijsc.2021.0330

PROTECTING VIRTUALIZED INFRASTRUCTURES IN CLOUD COMPUTING BASED ON BIG DATA SECURITY ANALYTICS

R.K. Monika and K. Ravikumar

Department of Computer Science and Engineering, Knowledge Institute of Technology, India

Abstract

Virtualized infrastructure in cloud computing has become an attractive target for cyber attackers to launch advanced attacks. This paper proposes a novel big data based security analytics approach to detecting advanced attacks in virtualized infrastructures. Network logs as well as user application logs collected periodically from the guest virtual machines (VMs) are stored in the Hadoop Distributed File System (HDFS). Then, extraction of attack features is performed through graph-based event correlation and Map Reduce parser based identification of potential attack paths. Next, determination of attack presence is performed through two-step machine learning, namely logistic regression is applied to calculate attack's conditional probabilities with respect to the attributes, and belief propagation is applied to calculate the belief in existence of an attack based on them. Experiments are conducted to evaluate the proposed approach using well-known malware as well as in comparison with existing security techniques for virtualized infrastructure. The results show that our proposed approach is effective in detecting attacks with minimal performance overhead.

Keywords:

Virtualized Infrastructure, Virtual Machines, Hadoop Distributed File System

1. INTRODUCTION

A virtualized infrastructure consists of virtual machines (VMs) that rely upon the software-defined multi-instance resources of the hosting hardware. The virtual machine monitor, also called hypervisor, sustains, regulates and manages the software-defined multi-instance architecture. The ability to pool different computing resources as well as enable on-demand resource scaling has led to the widespread deployment of virtualized infrastructures as an important provisioning to cloud computing services. This has made virtualized infrastructures become an attractive target for cyber attackers to launch attacks for illegal access. Exploiting the software vulnerabilities within the hypervisor source code, sophisticated attacks such as VENOM (Virtualized Environment Neglected Operations Manipulation) have been performed which allow an attacker to break out of a guest VM and access the underlying hypervisor

Many of today's devices are internet-enabled with IPv4 internet addresses, exposing them to internet threats. To determine the true scale of vulnerabilities being introduced, particularly in the IPv4 internet address space, a new methodology of scanning the entire IPv4 internet space is required. To improve scanning speeds we created a framework combining fast connectionless port scanners with a thorough and accurate connection-oriented scanner to verify results. The results are stored to a database. This combined framework provides more robust results than current connectionless scanners, yet still scans the IPv4 internet fast enough to be practically usable for mass scanning.

As the functional complexity of the malicious software increases, their analyses faces new problems. The paper presents these aspects in the context of automatic analyses of Internet threats observed with the Honey Pot technology. The problems were identified based on the experience gained from the analyses of exploits and malware using the dedicated infrastructure deployed in the network of the Institute of Computer Science at Warsaw University of Technology. They are discussed on the background of the real-life case of a recent worm targeting Network Attached Storage (NAS) devices vulnerability. The paper describes the methodology and data analysis supporting systems as well as the concept of general and custom Honey Pots used in the research.

Antivirus software is one of the most widely used tools for detecting and stopping malicious and unwanted files. However, the long term effectiveness of traditional host-based antivirus is questionable. Antivirus software fails to detect many modern threats and its increasing complexity has resulted in vulnerabilities that are being exploited by malware. This paper advocates a new model for malware detection on end hosts based on providing antivirus as an in-cloud network service. This model enables identification of malicious and unwanted software by multiple, heterogeneous detection engines in parallel, and a technique we term `N-version protection'. This approach provides several important benefits including better detection of malicious software, enhanced forensics capabilities, retrospective detection, and improved deploy ability and management. To explore this idea we construct and deploy a production quality in-cloud antivirus system called Cloud AV. Cloud AV includes a lightweight, cross-platform host agent and a network service with ten antivirus engines and two behavioral detection engines. We evaluate the performance, scalability, and efficacy of the system using data from a real-world deployment lasting more than six months and a database of 7220 malware samples covering a one year period. Using this dataset we find that Cloud AV provides 35% better detection coverage against recent threats compared to a single antivirus engine and a 98% detection rate across the full dataset. We show that the average length of time to detect new threats by an antivirus engine is 48 days and that retrospective detection can greatly minimize the impact of this delay. Finally, we relate two case studies demonstrating how the forensics capabilities of Cloud AV were used by operators during the deployment.

As the dominator of the Smartphone operating system market, consequently android has attracted the attention of s malware authors and researcher alike. The number of types of android malware is increasing rapidly regardless of the considerable number of proposed malware analysis systems. In this paper, by taking advantages of low false-positive rate of misuse detection and the ability of anomaly detection to detect zero-day malware, we propose a novel hybrid detection system based on a new open-source framework Cuckoo Droid, which enables the use of

Cuckoo Sandbox's features to analyze Android malware through dynamic and static analysis. Our proposed system mainly consists of two parts: anomaly detection engine performing abnormal apps detection through dynamic analysis; signature detection engine performing known malware detection and classification with the combination of static and dynamic analysis. We evaluate our system using 5560 malware samples and 6000 benign samples. Experiments show that our anomaly detection engine with dynamic analysis is capable of detecting zero-day malware with a low false negative rate (1.16 %) and acceptable false positive rate (1.30 %); it is worth noting that our signature detection engine with hybrid analysis can accurately classify malware samples with an average positive rate 98.94 %. Considering the intensive computing resources required by the static and dynamic analysis, our proposed detection system should be deployed off-device, such as in the Cloud. The app store markets and the ordinary users can access our detection system for malware detection through cloud service.

2. RELATED WORKS

Existing approaches to detecting attack presence are limited in terms of their ability to detect threats in real-time as well as to scale across multiple hosts. One of the limitations of existing security approaches stems from the use of a dedicated signature database for threat detection. This applies to approaches that feature a regularly updated attack signature database for threat detection. Typically in the in-VM and outside-VM interworking approach, an in-VM agent detects and passes any suspicious file to the remote scrutiny server, which uses the signature database to determine if it is a malware. The dependence on a regularly-updated signature database makes it limited in detecting zero-day attacks.

- *Volume*: Depending on the number of guest VMs and the size of the network, the amount of the network and user application logs to be collected can range from approximately 500 MB to 1 GB an hour;
- *Velocity*: The network and user application logs are collected in real-time, in order to detect the presence of malware and root kit attacks, accordingly the collected data containing its behavior needs to be processed as soon as possible;
- *Veracity*: Due to the "low and slow" approach that malware and root kit take in hiding their presence within the guest VMs, data analysis has to rely upon event correlation and advanced analytics.

3. PROPOSED SYSTEM

In an "attack pyramid" -based scheme is proposed to detect APTs (advanced persistent threats) in a large enterprise network environment. Based on threat tree modeling, different planes (namely hardware, user, network and application) to which an attack may be launched are placed hierarchically with the end goal placed at the top. First, outputs from all available sensors in the network (e.g. network logs, execution traces, etc.) are put into contexts. Then, in terms of the contexts various suspicious activities detected at each attack plane are correlated in a Map Reduce model, which takes in all the sensor outputs and generates an event set describing potential APTs. Finally, an alert system determines attack presence by calculating the confidence levels of each correlated event.

- The scheme first extracts malware features by using static as well as dynamic analysis on malware apps. The obtained features are then used to train a one-class SVM (Support Vector Machine) classifier for anomaly-based detection. Implemented on an emulated Android platform, Cuckoo Droid achieved a detection accuracy of 98.84 %.
- The obtained features are then used to train one-class SVM classifiers to detect malware presence within guest VMs.
- Implemented on KVM, the scheme is able to detect well-known DDoS (Distributed Denial of Service) and botnet attacks such as LOIC (Low Orbit Ion Cannon) and Zeus.

4. SOFTWARE DESCRIPTION

Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate. There's no language barrier with .NET: there are numerous languages available to the developer including Managed C++, C#, Visual Basic and Java Script. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate.

- ***Managed Code***: The code that targets .NET, and which contains certain extra information - "metadata" - to describe itself. Whilst both managed and unmanaged code can run in the runtime, only managed code contains the information that allows the CLR to guarantee, for instance, safe execution and interoperability.
- ***Managed Data***: With Managed Code comes Managed Data. CLR provides memory allocation and Deal location facilities, and garbage collection. Some .NET languages use Managed Data by default, such as C#, Visual Basic.NET and JScript.NET, whereas others, namely C++, do not. Targeting CLR can, depending on the language you're using, impose certain constraints on the features available. As with managed and unmanaged code, one can have both managed and unmanaged data in .NET applications - data that doesn't get garbage collected but instead is looked after by unmanaged code.
- ***Common Type System***: The CLR uses something called the Common Type System (CTS) to strictly enforce type-safety. This ensures that all classes are compatible with each other, by describing types in a common way. CTS define how types work within the runtime, which enables types in one language to interoperate with types in another language, including cross-language exception handling. As well as ensuring that types are only used in appropriate ways, the runtime also ensures that code doesn't attempt to access memory that hasn't been allocated to it.

The CLR provides built-in support for language interoperability. To ensure that you can develop managed code that can be fully used by developers using any programming

language, a set of language features and rules for using them called the Common Language Specification (CLS) has been defined. Components that follow these rules and expose only CLS features are considered CLS-compliant.

- ***Class Library***: NET provides a single-rooted hierarchy of classes, containing over 7000 types. The root of the namespace is called System; this contains basic types like Byte, Double, Boolean, and String, as well as Object. All objects derive from System. Object. As well as objects, there are value types. Value types can be allocated on the stack, which can provide useful flexibility. There are also efficient means of converting value types to object types if and when necessary.

The set of classes is pretty comprehensive, providing collections, file, screen, and network I/O, threading, and so on, as well as XML and database connectivity.

The class library is subdivided into a number of sets (or namespaces), each providing distinct areas of functionality, with dependencies between the namespaces kept to a minimum.

- ***Languages Supported By .Net***: The multi-language capability of the .NET Framework and Visual Studio .NET enables developers to use their existing programming skills to build all types of applications and XML Web services. The .NET framework supports new versions of Microsoft's old favorites Visual Basic and C++ (as VB.NET and Managed C++), but there are also a number of new additions to the family.

Visual Basic .NET has been updated to include many new and improved language features that make it a powerful object-oriented programming language. These features include inheritance, interfaces, and overloading, among others. Visual Basic also now supports structured exception handling, custom attributes and also supports multi-threading.

Visual Basic .NET is also CLS compliant, which means that any CLS-compliant language can use the classes, objects, and components you create in Visual Basic .NET.

Managed Extensions for C++ and attributed programming are just some of the enhancements made to the C++ language. Managed Extensions simplify the task of migrating existing C++ applications to the new .NET Framework.

C# is Microsoft's new language. It's a C-style language that is essentially "C++ for Rapid Application Development". Unlike other languages, its specification is just the grammar of the language. It has no standard library of its own, and instead has been designed with the intention of using the .NET libraries as its own.

Microsoft Visual J# .NET provides the easiest transition for Java-language developers into the world of XML Web Services and dramatically improves the interoperability of Java-language programs with existing software written in a variety of other programming languages.

Active State has created Visual Perl and Visual Python, which enable .NET-aware applications to be built in either Perl or Python. Both products can be integrated into the Visual Studio .NET environment. Visual Perl includes support for Active State's Perl Dev Kit.

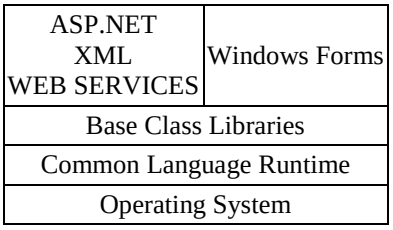

Fig.1. Net Framework

C#.NET is also compliant with CLS (Common Language Specification) and supports structured exception handling. CLS is set of rules and constructs that are supported by the CLR (Common Language Runtime). CLR is the runtime environment provided by the .NET Framework; it manages the execution of the code and also makes the development process easier by providing services.

C#.NET is a CLS-compliant language. Any objects, classes, or components that created in C#.NET can be used in any other CLS-compliant language. In addition, we can use objects, classes, and components created in other CLS-compliant languages in C#.NET .The use of CLS ensures complete interoperability among applications, regardless of the languages used to create the application.

- ***Constructors and Destructors***: Constructors are used to initialize objects, whereas destructors are used to destroy them. In other words, destructors are used to release the resources allocated to the object. In C#.NET the sub finalize procedure is available. The sub finalize procedure is used to complete the tasks that must be performed when an object is destroyed. The sub finalize procedure is called automatically when an object is destroyed. In addition, the sub finalize procedure can be called only from the class it belongs to or from derived classes.
- ***Garbage Collection***: Garbage Collection is another new feature in C#.NET. The .NET Framework monitors allocated resources, such as objects and variables. In addition, the .NET Framework automatically releases memory for reuse by destroying objects that are no longer in use.

In C#.NET, the garbage collector checks for the objects that are not currently in use by applications. When the garbage collector comes across an object that is marked for garbage collection, it releases the memory occupied by the object.

- ***Overloading***: Overloading is another feature in C#. Overloading enables us to define multiple procedures with the same name, where each procedure has a different set of arguments. Besides using overloading for procedures, we can use it for constructors and properties in a class.
- ***Multithreading***: C#.NET also supports multithreading. An application that supports multithreading can handle multiple tasks simultaneously, we can use multithreading to decrease the time taken by an application to respond to user interaction.
- ***Structured Exception Handling***: C#.NET supports structured handling, which enables us to detect and remove errors at runtime. In C#.NET, we need to use Try…Catch…Finally statements to create exception handlers. Using Try…Catch…Finally statements, we can

create robust and effective exception handlers to improve the performance of our application.

5. .NET FRAMEWORK

The .NET Framework is a new computing platform that simplifies application development in the highly distributed environment of the Internet.

- To provide a consistent object-oriented programming environment whether object codes is stored and executed locally on Internet-distributed, or executed remotely.
- To provide a code-execution environment to minimizes software deployment and guarantees safe execution of code.
- Eliminates the performance problems.

There are different types of application, such as Windows-based applications and Web-based applications.

5.1 FEATURES OF SQL-SERVER

The OLAP Services feature available in SQL Server version 7.0 is now called SQL Server 2000 Analysis Services. The term OLAP Services has been replaced with the term Analysis Services. Analysis Services also includes a new data mining component. The Repository component available in SQL Server version 7.0 is now called Microsoft SQL Server 2000 Meta Data Services. References to the component now use the term Meta Data Services. The term repository is used only in reference to the repository engine within Meta Data Services

5.2 SOFTWARE MODEL

The Waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement.

In this article, we will discuss the advantages and disadvantages of the waterfall, should we avoid it? when to use it? and the waterfall model pitfall, and why I see it as the father of the SDLC models.

Waterfall Model contains the main phases similarly to other process models, you can read this article for more information about phases definitions.

Due to the nature of the waterfall model, it is hard to get back to the previous phase once completed. Although, this is can be very rigid in some software projects which need some flexibility, while, this model can be essential or the most suitable model for other software projects' contexts.

The usage of the waterfall model can fall under the projects which do not focus on changing the requirements, for example:

1. Projects initiated from a request for proposal (RFP), the customer has a very clear documented requirements
2. Mission Critical projects, for example, in a Space shuttle
3. Embedded systems.

We can notice some similarities of these types of projects that they cannot be delivered in iterative, incremental, or agile manner, for example, in embedded systems for the elevator, you cannot deliver an elevator who can go up only without going down, or handling only users requests from inside and ignore outside calls for the elevator.

5.3 VALIDATION AND VERIFICATION MODEL: V-MODEL

V-Model is mostly known as the validation and verification software development process model (The Vee Model), and It is one of the most know software development methodology. Although it is considered as an improvement to the waterfall model and it has some similarities as the process also based on sequential steps moving down in a linear way, it differs from the waterfall model as the steps move upwards after the coding phase to form the typical V shape. This V shape demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.

5.4 V-MODEL

This means that any phase in the development process begins only if the previous phase is complete and has a correspondence related testing phase which is performed against this phase completion. Similar to the Waterfall model, the V-Model does not define the process to go back to the previous phase to handle changes in requirement.

The technical aspect of the project cycle is considered as a V shape starting with the business needs on the upper left and ending with the user acceptance testing on the upper right.

5.5 V-MODEL PHASES

The V-Model Model contains the main phases similarly to other process models, you can read this article for more information about SDLC phases definitions.

Moreover, it breaks down the testing phase into detailed steps to ensure the validation and verification process. So, it contains the below testing phases:

- **Unit Testing:** The Unit testing is the testing at the code level and helps eliminate issues at an early stage, mainly the developer is responsible to perform the unit test for his code while not all the defects cannot be discovered at the unit testing.
- **Functional Testing**: Functional testing is associated with the low-level design phase which ensures that collections of codes and units are working together probably to execute new function or service.

Table.2. Requirement Format

Req#	Requirement	Comments	Priority	SME Reviewed / Approved
BR_LR_05	The system should associate a supervisor indicator with each job class.	Business Process = "Maintenance	3	Bob Dylan, Mick Jagger
BR_LR_08	The system should handle any number of fees (existing and new) associated with unions.	Business Process = "Changing Dues in the System" An example of a new fee is an initiation fee.	2	Bob Dylan, Mick Jagger
BR_LR_10	The system should capture and maintain job class status (i.e., active or inactive)	Business Process = "Maintenance" Some job classes are old and are no longer used. However, they still need to be maintained for legal, contract and historical purposes.	2	Bob Dylan, Mick Jagger
BR_LR_16	The system should assign the Supervisor Code based on the value in the Job Class table and additional criteria as specified by the clients.	April 2005 – New requirement. It is one of three new requirements from BR_LR_03.	2	
BR_LR_18	The system should provide the Labor Relations office with the ability to override the system-derived Bargaining Unit code and the Union Code for to-be-determined employee types, including hourly appointments.	April 2005 – New requirement. It is one of three new requirements from BR_LR_04. 5/11/2005 – Priority changed from 2 to 3.	2 3	

Table.3. Deferred Requirements

Req#	Business Requirement	Status	Comments	Pri	SME Reviewed /Approved
BR_LR_01	The system should validate the relationship between Bargaining Unit/Location and Job Class.	April 2005: Deleted. This requirement has been replaced by BR_LR_036 and BR_CC_33.	Business Process = "Assigning a Bargaining Unit to an Appointment"	1	Bob Dylan, Mick Jagger
BR_LR_02	The system should validate that the supervisor indicator is correct according to job class. Deferred to Phase 2B: 3/29/2005	April 2005: Deferred to Phase 2B.	Business Process = "Assigning a Bargaining Unit to an Appointment"	3	Bob Dylan, Mick Jagger
BR_LR_03	The system should derive the bargaining unit code, union code, and supervisor indicator from the job class code and location.	April 2005: Deleted Replaced by BR_LR_16 and BR_LR_17.	Business Process = "Assigning a Bargaining Unit to an Appointment"; This will eliminate the need, typically, for the user to enter the bargaining unit code, union code and supervisor indicator.	1	Bob Dylan, Mick Jagger

- **Integration Testing**: Integration testing is associated with the high-level design phase. Integration testing ensures the integration between all system modules after adding any new functions or updates.
- **System Testing**: System testing is associated with the system requirements and design phase. It combines the software, hardware, and the integration of this system with the other external systems.
- **User Acceptance Testing**: User Acceptance testing is associated with the business and operations analysis phase. The customer users are the main performers of this testing based on test cases and scenarios that cover the business requirements to ensure that they have delivered the right software as per the specifications.

Identify any requirements that have been deleted after approval or that may be delayed until future versions of the system.

5.6 CASE STUDY

Computer-Aided Software Engineering (CASE) is the use of software tools to assist in the development and maintenance of software. Tools used to assist in this way are known as CASE Tools.

5.6.1 CASE Tool:

- A CASE tool is a computer-based product aimed at supporting one or more software engineering activities within a software development process.
- Computer-Aided Software Engineering tools are those software which are used in any and all phases of developing an information system, including analysis, design and programming. For example, data dictionaries and diagramming tools aid in the analysis and design phases, while application generators speed up the programming phase.
- CASE tools provide automated methods for designing and documenting traditional structured programming techniques. The ultimate goal of CASE is to provide a language for describing the overall system that is sufficient to generate all the necessary programs needed.

6. MODULE DESCRIPTION

- ***Hypervisor-Assisted Malware Detection***: Hypervisor-assisted malware detection, on the other hand, uses the underlying hypervisor to detect malware within the guest VMs. A hypervisor-assisted malware detection scheme is designed in to detect botnet activity within the guest VMs. The scheme installs a network sniffer on the hypervisor to monitor external traffic as well as inter-VM traffic. Implemented on Xen, it is able to detect the presence of the Zeus botnet on the guest VMs. A hypervisor-assisted detection scheme is proposed in using guest application and network flow characteristics.
- ***Clustering for Security Analytics***: Clustering organises data items in an unlabeled dataset into groups based on their feature similarities. For security analytics, clustering finds a pattern which generalises the characteristics of data items, ensuring that it is well generalized to detect unknown attacks. Examples of cluster based classifiers include K-means clustering and k-nearest neighbors, which are used in both intrusion detection and

Clustering is used for security analytics for industrial control systems in an NCI (networked critical infrastructure) environment. First, data outputs from various network sensors are arranged as vectors and K-means clustering is applied to group the vectors into clusters. The MapReduce model is then applied to the grouped clusters to find groupings of possible attack behaviour, thus allowing the detection to be carried out efficiently.

- ***Virtualized Environment Neglected Operations Manipulation***: A virtualized infrastructure consists of virtual machines (VMs) that rely upon the software-defined multi-instance resources of the hosting hardware. The virtual machine monitor, also called hypervisor, sustains, regulates and manages the software-defined multi-instance architecture. The ability to pool different computing resources as well as enable on-demand resource scaling has led to the widespread deployment of virtualized infrastructures as an important provisioning to cloud computing services. This has made virtualized infrastructures become an attractive target for cyber attackers to launch attacks for illegal access. Exploiting the software vulnerabilities within the hypervisor source code, sophisticated attacks such as VENOM (Virtualized Environment Neglected Operations Manipulation) have been performed which allow an attacker to break out of a guest VM and access the underlying hypervisor.
- ***Security Information and Event Management***: Security analytics applies analytics on the various logs which are obtained at different points within the network to determine attack presence. By leveraging the huge amounts of logs generated by various security systems (e.g., intrusion detection systems (IDS), security information and event management (SIEM), etc.), applying big data analytics will be able to detect attacks which are not discovered through signature- or rule-based detection methods. While security analytics removes the need for signature database by using event correlation to detect previously undiscovered attacks, this is often not carried out in real-time and current implementations are intrinsically non-scalable. To overcome these limitations, in this paper we propose a novel big data based security analytics (BDSA) approach to protecting virtualized infrastructures against advanced attacks.

7. SYSTEMS DESIGN

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap and synergy with the disciplines of systems analysis, systems architecture and systems engineering.

The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces. This section provides a brief description of the Systems Design Document's purpose and scope.

This section provides a description of the project from a management perspective and an overview of the framework within which the conceptual system design was prepared. If appropriate, include the information discussed in the subsequent sections in the summary.

This section describes the system in narrative form using non-technical terms. It should provide a high-level system architecture diagram showing a subsystem breakout of the system, if applicable. The high-level system architecture or subsystem diagrams should, if applicable, show interfaces to external systems. Supply a high-level context diagram for the system and subsystems, if applicable. Refer to the requirements trace ability matrix (RTM) in the Functional Requirements Document (FRD), to identify the allocation of the functional requirements into this design document.

8. SYSTEM ARCHITECTURE

In this section, describe the system and/or subsystem(s) architecture for the project. References to external entities should

be minimal, as they will be described in detail in Section 6, External Interfaces.

- ***System Hardware Architecture***: In this section, describe the overall system hardware and organization. Include a list of hardware components (with a brief description of each item) and diagrams showing the connectivity between the components. If appropriate, use subsections to address each subsystem.
- ***System Software Architecture***: In this section, describe the overall system software and organization. Include a list of software modules (this could include functions, subroutines, or classes), computer languages, and programming computer-aided software engineering tools (with a brief description of the function of each item). Use structured organization diagrams/object-oriented diagrams that show the various segmentation levels down to the lowest level. All features on the diagrams should have reference numbers and names. Include a narrative that expands on and enhances the understanding of the functional breakdown. If appropriate, use subsections to address each module.
- ***Internal Communications Architecture***: In this section, describe the overall communications within the system; for example, LANs, buses, etc. Include the communications architecture(s) being implemented, such as X.25, Token Ring, etc. Provide a diagram depicting the communications path(s) between the system and subsystem modules. If appropriate, use subsections to address each architecture being employed.
- ***File and Database Design***: Interact with the Database Administrator (DBA) when preparing this section. The section should reveal the final design of all database management system (DBMS) files and the non-DBMS files associated with the system under development. Additional information may add as required for the particular project. Provide a comprehensive data dictionary showing data element name, type, length, source, validation rules, maintenance (create, read, update, delete (CRUD) capability), data stores, outputs, aliases, and description. Can be included as an appendix.

8.1. DETAILED DESIGN

This section provides the information needed for a system development team to actually build and integrate the hardware components, code and integrate the software modules, and interconnect the hardware and software segments into a functional product. Additionally, this section addresses the detailed procedures for combining separate COTS packages into a single system. Every detailed requirement should map back to the FRD, and the mapping should be presented in an update to the RTM and include the RTM as an appendix to this design document.

8.2. HARDWARE DETAILED DESIGN

A hardware component is the lowest level of design granularity in the system. Depending on the design requirements, there may be one or more components per system. This section should provide enough detailed information about individual component requirements to correctly build and/or procure all the hardware for the system (or integrate COTS items).

If there are many components or if the component documentation is extensive, place it in an appendix or reference a separate document. Add additional diagrams and information, if necessary, to describe each component and its functions, adequately. Industry-standard component specification practices should be followed. For COTS procurements, if a specific vendor has been identified, include appropriate item names.

8.3. SOFTWARE DETAILED DESIGN

A software module is the lowest level of design granularity in the system. Depending on the software development approach, there may be one or more modules per system. This section should provide enough detailed information about logic and data necessary to completely write source code for all modules in the system (and/or integrate COTS software programs).

If there are many modules or if the module documentation is extensive, place it in an appendix or reference a separate document. Add additional diagrams and information, if necessary, to describe each module, its functionality, and its hierarchy. Industry-standard module specification practices should be followed. Include the following information in the detailed module designs:

If the system includes more than one component there may be a requirement for internal communications to exchange information, provide commands, or support input/output functions. This section should provide enough detailed information about the communication requirements to correctly build and/or procure the communications components for the system. Include the following information in the detailed designs (as appropriate):

8.4. EXTERNAL INTERFACES

External systems are any systems that are not within the scope of the system under development, regardless whether the other systems are managed by the State or another agency. In this section, describe the electronic interface(s) between this system and each of the other systems and/or subsystem(s), emphasizing the point of view of the system being developed.

In this section, describe the interface(s) between the system being developed and other systems; for example, batch transfers, queries, etc. Include the interface architecture(s) being implemented, such as wide area networks, gateways, etc. If appropriate, use subsections to address each interface being implemented.

For each system that provides information exchange with the system under development, there is a requirement for rules governing the interface. This section should provide enough detailed information about the interface requirements to correctly format, transmit, and/or receive data across the interface. Include the following information in the detailed design for each interface (as appropriate):

8.5. INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written

or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
- It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
- When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

8.6. OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

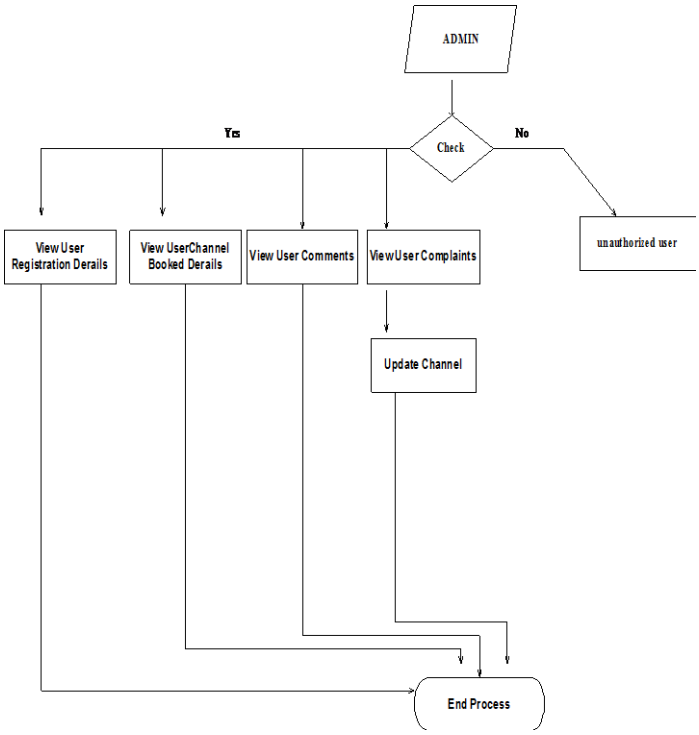

Fig.1. System Design

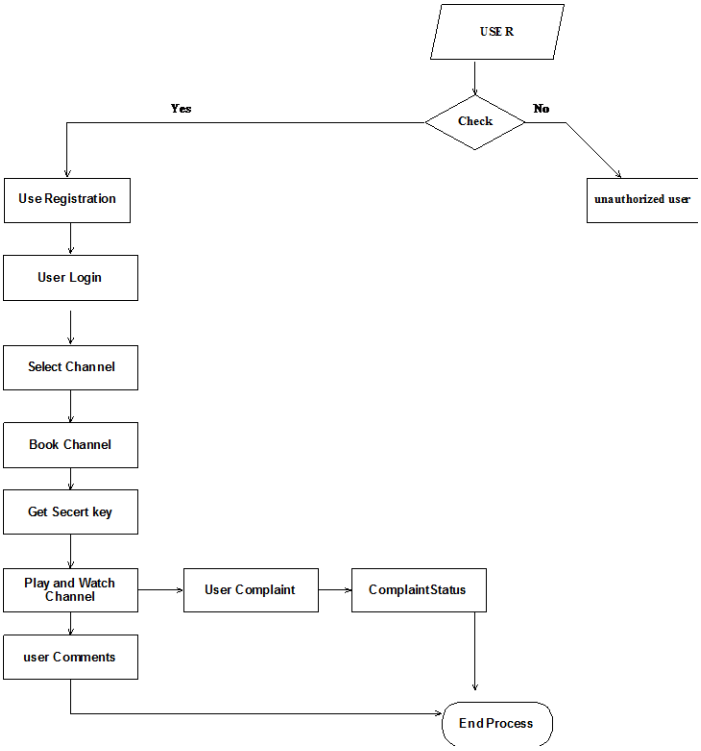

Fig.2. User Case Diagram

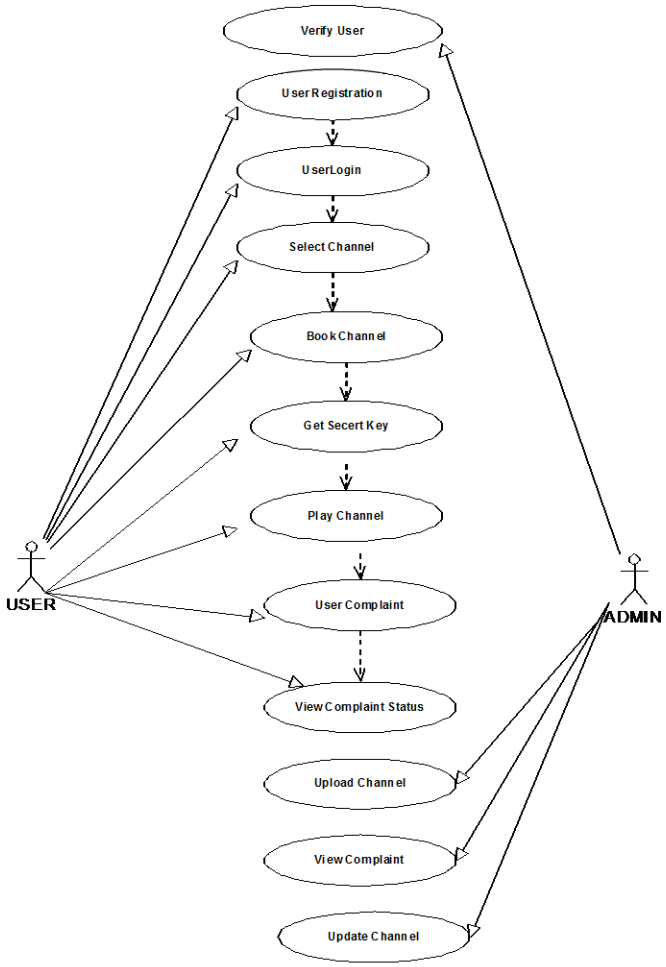

Fig.3. Class Diagram

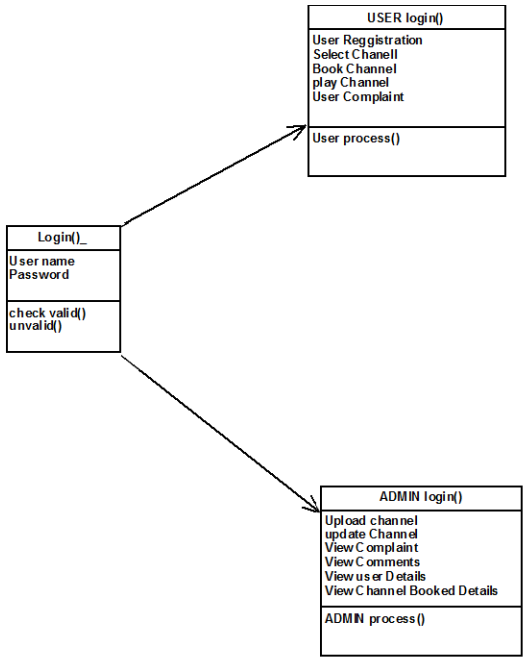

Fig.4. Activity Diagram

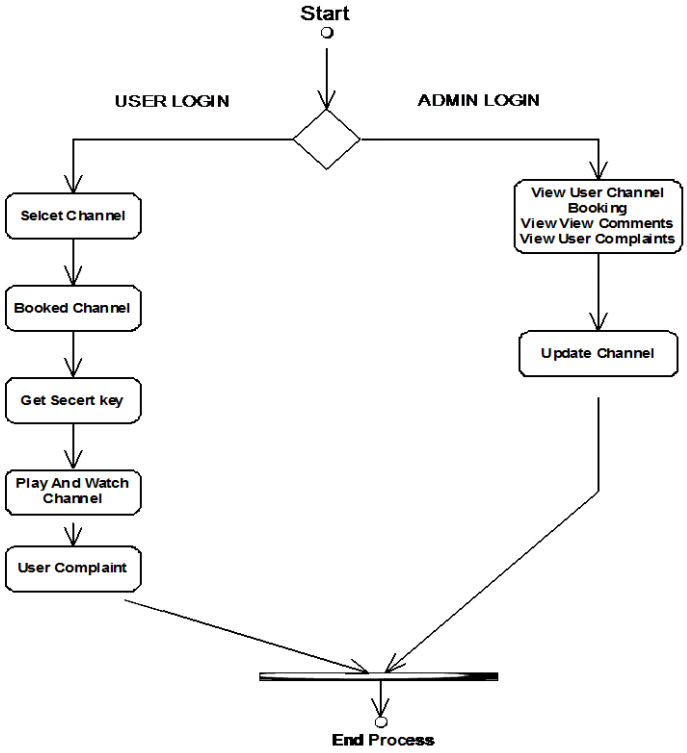

Fig.5. Sequence Diagram

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system.

8.7. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

9. CONCLUSION

In this paper, we have put forward a novel big data based security analytics (BDSA) approach to protecting virtualized infrastructures in cloud computing against advanced attacks. Our BDSA approach constitutes a three phase framework for detecting advanced attacks in real-time. First, the guest VMs network logs as well as user application logs are periodically collected from the guest VMs and stored in the HDFS. Then, attack features are extracted through correlation graph and MapReduce parser. Finally, two-step machine learning is utilized to ascertain attack presence. Logistic regression is applied to calculate attack's conditional probabilities with respect to individual attributes. Furthermore, belief propagation is applied to calculate the overall belief of an attack presence. From the second phase to the third, the extraction of attack features is further strengthened towards the determination of attack presence by the two-step machine learning. The use of logistic regression enables the fast calculation of attack's conditional probabilities.

The effectiveness of our BDSA approach is evaluated by testing it against well-known malware and rookit attacks. In all cases, it has been shown that our BDSA approach is able to detect them while maintaining a consistent performance overhead with increasing number of guest VMs at an average detection time of approximately 0.06 ms. Tested against Livewire, our BDSA approach incurs less performance overhead in attack detection through monitoring the guest VM's behavior. Our BDSA approach has taken advantage of the distributed processing of HDFS and real-time ability of Map Reduce model in Spark to address the velocity and volume challenges in security analytics.

REFERENCES

[1] D. Fisher, "Venom' Flaw in Virtualization Software Could Lead to VM Escapes, Data Theft", Available at: https://threatpost.com/venomflaw- in-virtualization-software-could-lead-to-vm-escapes-datatheft/ 112772/, 2015, Accessed at 2015.

[2] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N.Weaver, J. Amann, J. Beekman and M. Payer, "The matter of Heart Bleed", *Proceedings of International Conference on Internet Measurement*, pp. 475-488, 2014.

[3] K. Cabaj, K. Grochowski and P. Gawkowski, "Practical Problems of Internet Threats Analyses", *Proceedings of*

International Conference on Theory and Engineering of Complex Systems and Dependability, pp. 87-96, 2015.

[4] J. Oberheide, E. Cooke and F. Jahanian, "Cloudav: N-Version Antivirus in the Network Cloud", *Proceedings of International Symposium on USENIX Security*, pp. 91-106, 2008.

[5] X. Wang, Y. Yang and Y. Zeng, "Accurate Mobile Malware Detection and Classification in the Cloud", *Springer Plus*, Vol. 4, No. 1, pp. 1-23, 2015**.**

[6] N.V. Kousik, S. Jayasr and A. Daniel, "A Survey on Various Load Balancing Algorithm to Improve the Task Scheduling in Cloud Computing Environment", *Journal of Advanced Research in Dynamical and Control Systems*, Vol. 11, No. 8, pp. 2397-2406, 2019.

[7] V. Ganesan and S.G. Dhas, "Analysis on Improving the Response Time with PIDSARSA-RAL in Clowd Flows Mining Platform", *EAI Endorsed Transactions on Energy Web*, Vol. 5, No. 20, pp. 1-14, 2018.

[8] K. Cabaj, K. Grochowski and P. Gawkowski, "Practical Problems of Internet Threats Analyses", *Proceedings of International Conference on Dependability and Complex Systems*, pp. 87-96, 2015.

[9] Y. Lee and D. Kim, "Threats Analysis, Requirements and Considerations for secure Internet of Things", *International Journal of Smart Home*, Vol. 9, No. 12, pp. 191-198, 2015.

[10] S. Fedushko and E. Benova, "Semantic Analysis for Information and Communication Threats Detection of Online Service Users", *Procedia Computer Science*, Vol. 160, pp. 254-259, 2019.

[11] Y. Lee, Y. Park and D. Kim, "Security Threats Analysis and Considerations for Internet of Things", *Proceedings of International Conference on Security Technology*, pp. 28-30, 2015.

[12] R.E. Crossler, F. Belanger and D. Ormond, "The Quest for Complete Security: An Empirical Analysis of Users' Multi-Layered Protection from Security Threats", *Information Systems Frontiers*, Vol. 21, no. 2, pp. 343-357, 2019.