ISSN 1512-1232

Distributed Intrusion Detection System Scalability Enhancement using Cloud Computing

Manish Kumar
manishkumarjsr@yahoo.com

Abstract

Computer security is defined as the protection of computing systems against threats to confidentiality, integrity, and availability. An intrusion is defined as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource. An intrusion detection system (IDS) monitors network traffic and monitors for suspicious activity and alerts the system or network administrator. It identifies unauthorized use, misuse, and abuse of computer systems by both system insiders and external penetrators. Intrusion detection systems (IDS) are essential components in a secure network environment, allowing for early detection of malicious activities and attacks. By employing information provided by IDS, it is possible to apply appropriate countermeasures and mitigate attacks that would otherwise seriously undermine network security. However, current high volumes of network traffic overwhelm most IDS techniques requiring new approaches that are able to handle huge volume of log and packet analysis while still maintaining high throughput. A Distributed IDS (dIDS) consists of several IDS over a large network (s), all of which communicate with each other, or with a central server that facilitates advanced network monitoring. In a distributed environment, dIDS are implemented using cooperative intelligent agents distributed across the network(s). We propose an architecture for distributed Network Intrusion Detection System where comprehensive data analysis is executed in a cloud computing environment. The proposed architecture is able to efficiently handle large volumes of collected data and consequent high processing loads using cloud computing infrastructure. The main focus of the paper is to enhance the throughput and scalability of the dIDS.

Keywords*. Distributed Intrusion Detection System, Hadoop File System, Cloud Computing, MapReduce.*

I. Introduction

As information systems have become more comprehensive and a higher value asset of organizations, intrusion detection systems have been incorporated as elements of operating systems and network. Intrusion detection involves determining that some entity, an intruder, has attempted to gain, or worse, has gained unauthorized access to the system.

Intrusion detection systems (IDS) have a few basic objectives. Among these objectives are Confidentiality, Integrity, Availability, and Accountability.

Intrusion detection has traditionally been performed at the operating system (OS) level mostly by comparing expected and observed system resource usage. OS intrusion detection systems (OS IDS) can only detect intruders, internal or external, who perform specific system actions in a specific sequence known to constitute an intrusion or those intruders whose behavior pattern statistically varies from a norm. Internal intruders are said to comprise at least fifty percent of intruders, but OS intrusion detection systems are frequently insufficient to catch such intruders since they neither perform the specific intrusive actions because they are already legitimate users of the system, nor significantly deviate from expected behavior.

The most popular way to detect intrusions has been by using the audit data generated by the network or operating system. An audit trail is a record of activities on a system that are logged to a file in chronologically sorted order. Since almost all activities are logged on , it is possible that a manual inspection of these logs would allow intrusions to be detected. However, the incredibly large sizes of audit data generated (on the order of 100 Megabytes a day) make manual analysis impossible. IDSs automate the drudgery of wading through the audit data jungle. Audit trails are particularly useful because they can be used to establish guilt of attackers, and they are often the only way to detect unauthorized but subversive user activity.

Many times, even after an attack has occurred, it is important to analyze the audit data so that the extent of damage can be determined, the tracking down of the attackers is facilitated, and steps may be taken to prevent such attacks in future. An IDS can also be used to analyze audit data for such insights. This makes IDSs valuable as real-time as well as post-mortem analysis tools.

Intrusion Detection Systems (IDS) are important mechanisms which play a key role in network security and self-defending networks. Such systems perform automatic detection of intrusion attempts and malicious activities in a network through the analysis of traffic captures and collected data in general. Such data is aggregated, analyzed and compared to a set of rules in order to identify attack signatures, which are traffic patterns present in captured traffic or security logs that are generated by specific types of attacks. In the process of identifying attacks and malicious activities an IDS parses large quantities of data searching for patterns which match the rules stored in its signature database. Such procedure demands high processing power and data storage access velocities in order to be executed efficiently in large networks.

A. *Classification of Intrusion Detection Systems*

Intrusions can be divided into 6 main types:-

1. Attempted break-ins, which are detected by a typical behavior profiles or violations of security constraints.
2. Masquerade attacks, which are detected by atypical behavior profiles or violations of security constraints.
3. Penetration of the security control system, which are detected by monitoring for specific patterns of activity.
4. Leakage, which is detected by atypical use of system resources.
5. Denial of service, which is detected by atypical use of system resources.
6. Malicious use, which is detected by atypical behavior profiles, violations of security constraints, or use of special privileges.

However, we can divide the techniques of intrusion detection into two main types. IDSs issue security alerts when an intrusion or suspect activity is detected through the analysis of different aspects of collected data (e.g. packet capture files and system logs). Classical intrusion detection systems are based on a set of attack signatures and filtering rules which model the network activity generated by known attacks and intrusion attempts 357. Intrusion detection systems detect malicious activities through basically two approaches: anomaly detection and signature detection.

i. *Anomaly Detection*

This technique is based on the detection of traffic anomalies. The deviation of the monitored traffic from the normal profile is measured. Various different implementations of this technique have been proposed, based on the metrics used for measuring traffic profile deviation.

Anomaly detection techniques assume that all intrusive activities are necessarily anomalous. This means that if we could establish a "normal activity profile" for a system, we could, in theory, flag all system states varying from the established profile by statistically significant amounts as intrusion attempts. However, if we consider that the set of intrusive activities only intersects the set of anomalous activities instead of being exactly the same, we find a couple of interesting possibilities: (1) Anomalous activities that are not intrusive are flagged as intrusive. (2) Intrusive activities that are not anomalous result in false negatives (events are not flagged intrusive, though they actually are). This is a dangerous problem, and is far more serious than the problem of false positives.

The main issues in anomaly detection systems thus become the selection of threshold levels so that neither of the above 2 problems is unreasonably magnified, and the selection of features to monitor. Anomaly detection systems are also computationally expensive because of the overhead of keeping track of, and possibly updating several system profile metrics. Some systems based on this technique are discussed in Section 4 while a block diagram of a typical anomaly detection system is shown in Fig 1.

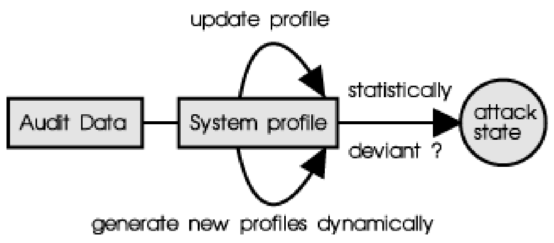

Fig 1:- IDS Anomaly Detection System

ii. *Misuse Detection*

This technique looks for patterns and signatures of already known attacks in the network traffic. A constantly updated database is usually used to store the signatures of known attacks. The way this technique deals with intrusion detection resembles the way that anti-virus software operates.

The concept behind misuse detection schemes is that there are ways to represent attacks in the form of a pattern or a signature so that even variations of the same attack can be detected. This means that these systems are not unlike virus detection systems -- they can detect many or all known attack patterns, but they are of little use for as yet unknown attack methods. An interesting point to note is that anomaly detection systems try to detect the complement of "bad" behavior. Misuse detection systems try to recognize known "bad" behavior. The main issues in misuse detection systems are how to write a signature that encompasses all possible variations of the pertinent attack, and how to write signatures that do not also match non-intrusive activity. Several methods of misuse detection, including a new pattern matching model are discussed later. A block diagram of a typical misuse detection system is shown in Fig 2 below.

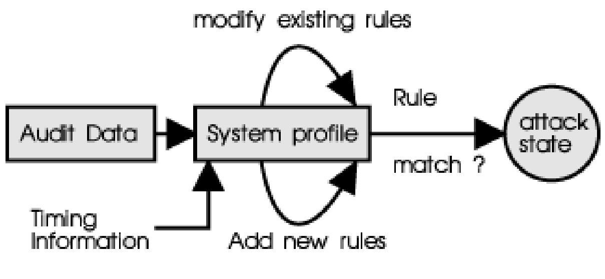

Fig 2:-IDS Misuse Detection System

Intrusion detection systems are further can also be classified in two groups, Network Intrusion Detection Systems (NIDS), which are based on data collected directly from the network, and Host Intrusion Detection Systems (HIDS), which are based on data collected from individual hosts. HIDSs are composed basically by software agents which analyze application and operating system logs, file system activities, local databases and other local data sources, reliably identifying local intrusion attempts. Such systems are not affected by switched network environments (which segment traffic flows) and is effective in environments where network packets are encrypted (thwarting usual traffic analysis techniques). However, they demand high processing power overloading the nodes' resources and may be affected by denial-of-service attacks. In face of the growing volume of network traffic and high transmission rates, software based NIDSs present performance issues, not being able to analyses all the captured packets rapidly enough. Some hardware based NIDSs offer the necessary analysis throughput but the cost of such systems is too high in relation to software based alternatives.

From the above, it is clear that as IDS grow in function and evolve in power, they also evolve in complexity. Agents of each new generation of IDS use agents of the previous generation as data sources, applying ever more sophisticated detection algorithms to determine ever more targeted responses. Often, one or more IDS and management system(s) may be deployed by an organization within its own network, with little regard to their neighbors or the global Internet. Just as all individual networks and intranets connect to form "The Internet", so can information from stand-alone internal and perimeter host- and network-based intrusion detection systems be combined to create a distributed Intrusion Detection System (dIDS).

Current IDS technology is increasingly unable to protect the global information infrastructure due to several problems:

i. The existence of single intruder attacks that cannot be detected based on the observations of only a single site.
ii. Coordinated attacks involving multiple attackers that require global scope for assessment.
iii. Normal variations in system behavior and changes in attack behavior that cause false detection and identification.
iv. detection of attack intention and trending is needed for prevention
v. Advances in automated and autonomous attacks, i.e. rapidly spreading worms, require rapid assessment and mitigation, and
vi. The sheer volume of attack notifications received by ISPs and host owners can become overwhelming.
vii. If aggregated attack details are provided to the responsible party, the likelihood of a positive response increases.

II. Distributed Intrusion Detection System

A distributed IDS (dIDS) consists of multiple Intrusion Detection Systems (IDS) over a large

A number of IDSs have been proposed for a networked or distributed environment. Early systems included ASAX 1, DIDS14 and NSTAT 13. These systems require the audit data collected from different places to be sent to a central location for an analysis.

The scalability of such systems is limited due to their centralized nature. To improve scalability later systems such as EMERALD 12, GriDS 15 and AAFID 4, deployed instruction detection systems at different locations and organized them into a hierarchy such that low-level IDSs send designated information to higher level IDSs.

However, the main problem with such an approach is that if two or more IDSs that are far apart in the hierarchy detect a common intruder, the two detection cannot be correlated until the messages from the different IDSs reach a common high-level IDS. This will require the messages to traverse multiple IDSs resulting in communication overheads. The Common Intrusion Detection Framework (CIDF) goes one step further as it aims to enable different intrusion detection and response components to interoperate and share information and resources in a distributed environment.

A. Advantages of a dIDS

Due to the greater view the agent allows the analyst to achieve, the dIDS offers the incident analyst many advantages over other single mode IDS systems. One of these advantages is the ability to detect attack patterns across an entire corporate network, with geographic locations separating segments by time zones or even continents. This could allow for the early detection of a well-planned and coordinated attack against the organization in question, which would allow the security people to ensure that targeted systems are secured and offending IPs are disallowed any access. Another proven advantage is to allow early detection of an Internet worm making its way through a corporate network. This information could then be used to identify and clean systems that have been infected by the worm, and prevent further spread of the worm into the network, therefore lowering any financial losses that would otherwise have been incurred.

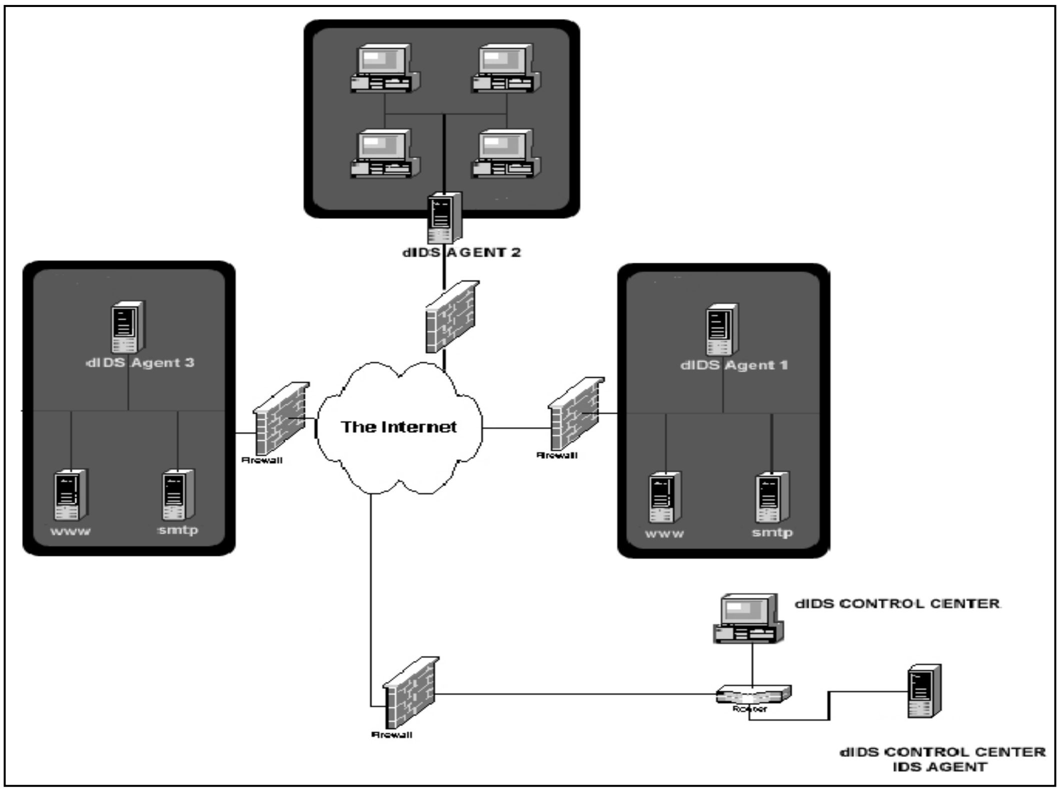

Fig 3:- Distributed IDS Architecture

As attackers, and attack methods become increasingly complex, the need for a dIDS system in large corporate, and military networks increases drastically. With the increased complexity of these attacks, analysts are leaving themselves open to the problems of communications breakdowns, where one analyst sees a single attack on his segment, and dismisses it as nothing. While several other segments receiving the same attacks in a coordinated manner, their analysts may be dismissing the seriousness of the attack. However, when all the attack data is viewed together, a dramatically different perspective the attack may emerge.

The dIDS system gives the analyst a quicker, easier, more efficient method to identify coordinated attacks across multiple network segments, and to trace back the activities of the attackers. The system also, ultimately, saves the corporation whose networks it is deployed on money by reducing the number of Incident Analysts needed, as well as the amount of time required to gather logs from the various IDS systems setup in a large corporate network. By having all of these attack records stored in a single place, it allows the analyst much more flexibility in discovering attack patterns, and other attack issues which may have otherwise gone unnoticed.

B. Incident Analysis With dIDS

Incident analysis using the dIDS system is really what it is all about. This is where all the power, potential, flexibility, and strength of the system as a whole lies. It is the reason why the dIDS was first conceptualized, to allow for advanced analysis of attacks occurring over multiple network segments, and at an advanced level11.

Aggregation is one of the common approach which is used to facilitate this advanced method of analysis across a networks multiple segments. By aggregating similar or related data, the analyst is able to easily see how an attack progressed through the different stages: from active network reconnaissance, to the final attack. It is possible for the incident analyst to see what kind of time frame the attacker was working within and to correlate other attack attempts against the networks to determine if there were multiple co-operative attackers. The most common methods of aggregation are according to attacker IP, destination port, agent ID, date, time, protocol, or attack type.

- Aggregating by attacker IP allows the analyst to view the steps of an attacker’s attempt from start to finish across the multiple network segments.
- Aggregating by destination port allows an analyst to view new trends in attack types, and to be able to identify new attack methods, or exploits being used.
- Aggregating by agent ID allows an analyst to see what variety of attacks and attackers have made attempts on the specific network segment the agent is on. Consequently, the analyst can determine if there are multiple attackers working in conjunction, or if there are network segments that are of more interest to attackers than others, thereby giving the security team a list of common targets to work on.
- Aggregating by date and time allows the analyst to view new attack patterns, and to potentially identify new worms or viruses that are only triggered at certain times.
- Aggregating by protocol helps in a purely statistical manner, which could allow an analyst to identify new attacks in particular protocols, or identify protocols on a network segment that should, under no circumstances, be there anyhow.
- Aggregating by attack type also allows for attack pattern matching and to correlate coordinated attacks against multiple network segments.

By utilizing all of these aggregation methods, the analyst is given an unlimited number of different sets of data to correlate against other attacks, detect coordinated distributed attacks, attacks from within their own network, and to detect new exploits and vulnerabilities being deployed by the underground hacking community.

The broad view given by the dIDS system also allows the analyst to ensure a minimum of false positives and false negatives by being able to see beyond a single network segment, into the network as a whole. For example, if the analyst saw that one out of five network segments got seven unrequested ICMP Echo packets, it could be a simple issue of false addressing or improper routing somewhere. However, if the analyst were to see that three separate network segments were reporting seven unrequested ICMP Echo packets, it is much more likely that these packets would be malicious in nature. This would cause the analyst to take note of the activity and perhaps check into the incident further or flag it for review at a later date1611.

III. Cloud Computing And The Map Reduce Framework

In this section we discuss the basic concepts of cloud computing and the MapReduce framework, introducing the architecture of the MapReduce implementation provided by the Hadoop project. The constant and rapid growth in the volume of data and communications in current networks and information systems raises the need for efficient scalable data storage and processing techniques. In order to address this issue, a new paradigm commonly called cloud computing was introduced. The main characteristic of this paradigm is the decentralization of data processing and storage through clustered environments that seamlessly scale to fit the constantly increasing demand, offering high performance and achieving efficient response times.

Usually a cloud computing environment is composed by a data processing cluster coupled with a storage area network to provide easily scalable resources. The users and applications access this environment seamlessly and transparently through standard API frontends, eliminating the need for knowledge of the specific network infrastructure and underlying services. Such architectures enable fast development and provisioning of large scale applications that handle a high number of requests or large quantities of data. Applications running on a cloud environment are able to scale transparently and automatically by simply adding more cloud resources, which are handled by the corresponding API.

A cloud application can be easily developed by utilizing the proper frontend API calls to perform the desired actions on the cloud environment (e.g. store and retrieve files or sort data). After an application is functional, it can be transparently scaled by adding more cloud storage or processing resources. Hence, cloud applications can potentially scale to handle any quantity of data. These characteristics make such platforms ideal for constructing efficient and scalable distributed intrusion detection data analysis systems.

This approach makes applications extremely flexible, since MapReduce allows for dynamic configuration of the quantity of maps and reduces. In other words, it is possible to configure the granularity of the data splitting process, thus adjusting the size of transitory key/value pairs that are distributed among the processing nodes in the cloud.

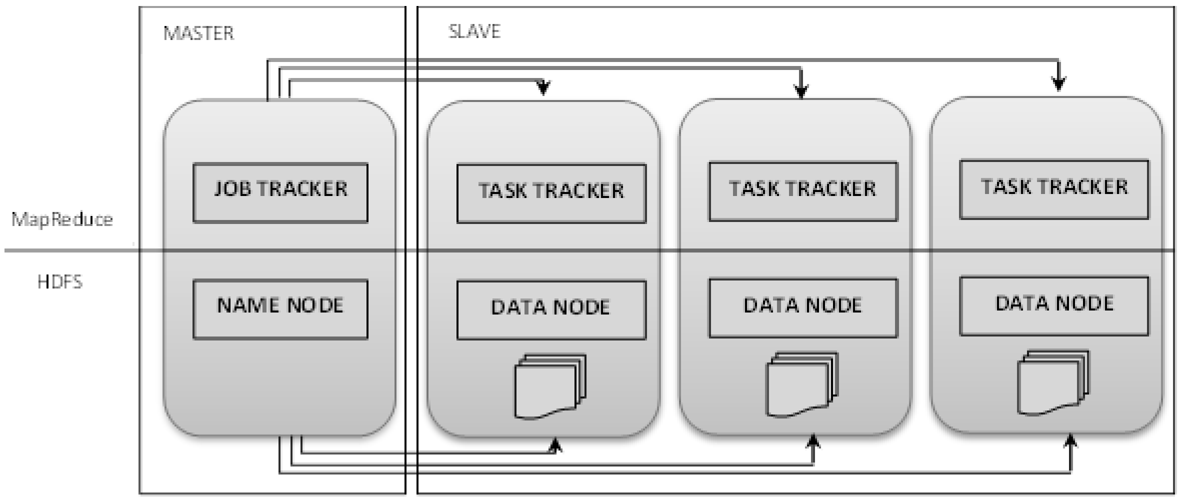

Fig 4:- *Hadoop Cluster Structure*

A. The Hadoop MapReduce Architecture

Hadoop is an open-source distributed framework written in Java programming language under Apache License 269 1017. It supports functions for processing huge size data. Hadoop system is constructed with a master server and multiple slave nodes as shown in Fig. 4. Hadoop has two

MapReduce component in Hadoop controls job processing in distributed manner. JobTracker process in the master server allocates jobs to TaskTrackers in slave nodes and merges results received from the TaskTrackers. When a task process a file in HDFS, each block of the file can be processed by an allocated task in DataNode in each slave node. In order to process the file, jobs are allocated to the slave nodes using map function provided by Hodoop. Result of the jobs are transmitted to the node where reduce function runs as shown in Fig. 5. Reduce function arranges the result and generates a result file. The number of nodes that run reduce function is determined by the number of blocks in the file and the number of reduce functions is decided by user parameter.

IV. Distributed IDS Integration with Cloud Infrastructure

In this section, we introduce the proposed architecture for scalable distributed intrusion detection systems using cloud infrastructure. The goal of this distributed intrusion detection system architecture is for solving two main issues: comprehensive data collection across different network areas and scalable efficient processing and analysis of the resulting dataset. In order to achieve this, we combine distributed data collection approaches with cloud computing techniques, achieving a hybrid system where data collection is performed by software sensor agents installed in different host in the network and data processing is carried out at MapReduce cloud that aggregates and analyses collected data. The cloud based attack detection application is able to correlate diverse collected data (such as traffic captures and logs) in order to detect complex attacks and it may easily scale through the addition of more worker nodes, being easily adaptable to different networks.

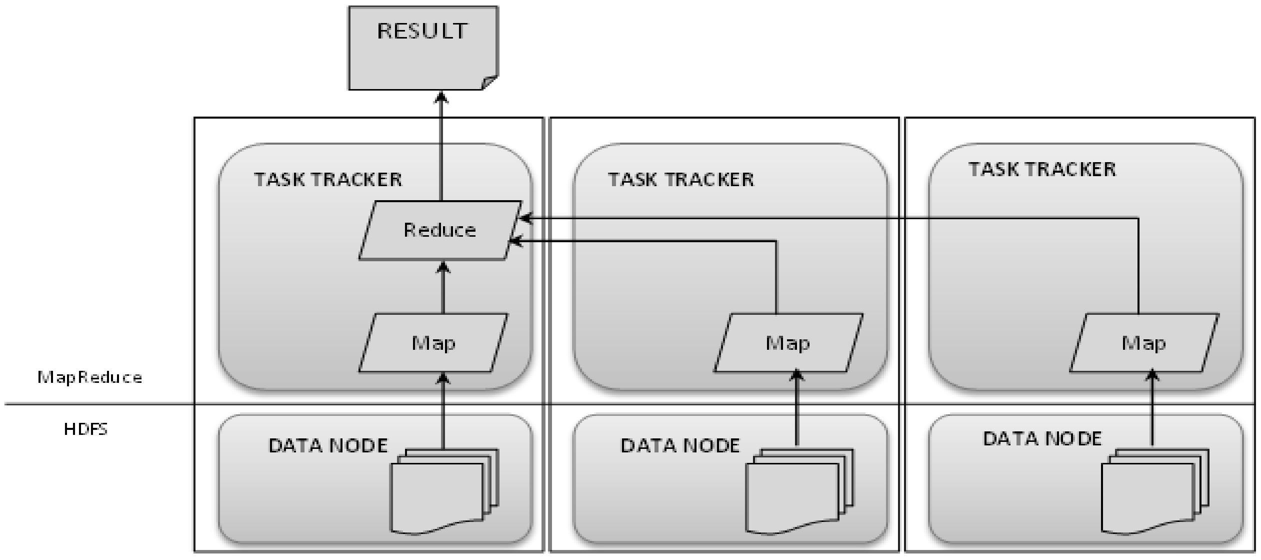

Fig 5:- *Hadoop MapReduce Function*

The proposed distributed IDS architecture is composed by mainly two parts: the sensor agents, the cloud based data storage and processing infrastructure. In this architecture, information flows from the data collection sensor agents installed in different network areas to the central MapReduce cloud. Differently from other approaches, instead of being processed by a centralized system, the collected data is then analysed by a cloud application that leverages the resources of the worker nodes in the cloud, scaling transparently as network traffic (and consequently the analysis dataset) grows. The proposed architecture is depicted in Fig 6.

The architecture provides an efficient topology for distributed data collection and storage, tasks that are of great importance in handling large quantities of distributed data (such as logs in the distributed intrusion detection system). It builds on the fact that the Hadoop implementation of MapReduce framework provides both a distributed filesystem for data storage and a data processing environment. The logs and traffic collected from the different systems spread in the network are aggregated in distributed file system volumes, which may be subsequently accessed by MapReduce applications running on the cloud infrastructure. Notice that this increase the storage capacity of the data processing environment while decreasing data access times, since the data is already in the cluster.

In our proposed architecture Hadoop environment aggregate data received from the individual sensors and process it through attack detection algorithm. This is an heterogeneous environment composed of different computers with different resources and architectures. In theory, any platform capable of running a Hadoop MapReduce framework implementation can be used in the cloud infrastructure to process IDS. Since the only requirement for running Hadoop is an Java Virtual Machine (JVM), this flexibility makes it viable to use legacy equipment for dIDS log analysis, reducing the costs of implementing such systems.

The hosts in the MapReduce cloud are also part of a distributed filesystem where the data collected by the sensor agents is stored during analysis. The cloud's master node receives the data and stores it in the distributed filesystem where it is accessed and modified in the analysis process. The distributed filesystem seamlessly scales together with the cloud infrastructure providing enough storage space to large quantities of logs without requiring special storage devices. Moreover, filesystems access speed is improved by distributing data among the cloud needs.

Several intrusion detection algorithms, data analysis, sensor fusion and event correlation models are intended to run as MapReduce jobs on the cloud infrastructure, which provides scalable performance for increasingly large volumes of data processing tasks. Information such as network flows (obtainable from packet capture files) is efficiently processed in a MapReduce grid, yielding fast results even in settings with sheer quantities of logs. This system can also be used to calculate statistical data regarding network activities and monitored nodes security.

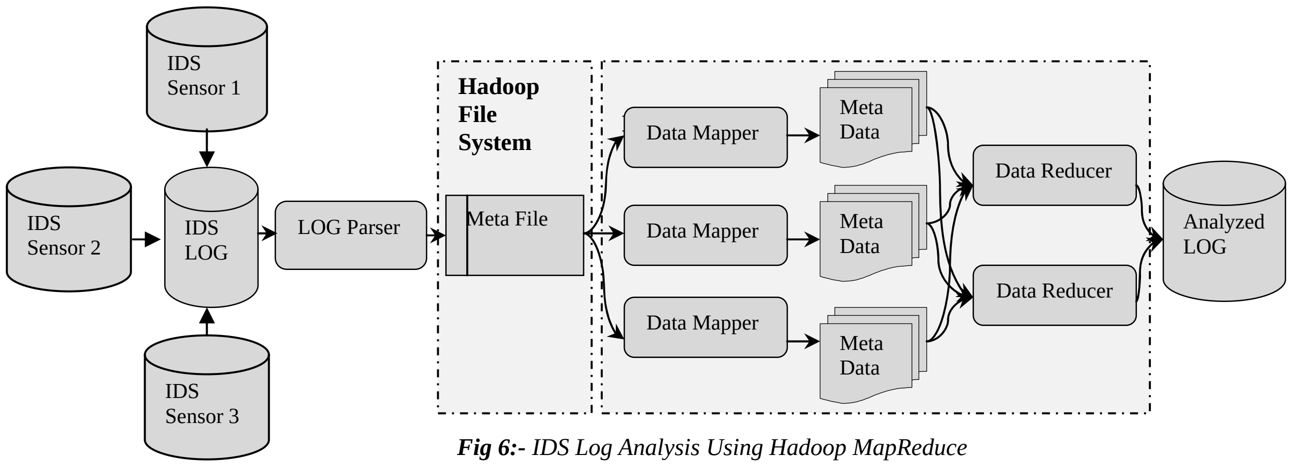

Fig 6:- *IDS Log Analysis Using Hadoop MapReduce*

We are implementing the dIDS log analysis using the cloud architecture with the help of Hadoop and MapReduce. The log file is produced by dIDS. Regular Parser, Analysis Procedure, Data Mapper, Data Reducer are processes component of our cloud based dIDS log analysis architecture as shown in Fig 6. Meta file is a file transferred into distributed file system. All of metadata are intermediate product of Job Dispatcher between Data Mapper and Data Reducer. At the beginning of procedure, alert generator collects malicious packets and stores information into a log file. The first component, Regular Parser, extracts the essential information and discards useless data then parses as a regular form. Next step, system would transfers the metafile to distributed file system which splits metafile spread every node. Job Dispatcher launches Data Mapper and assigns jobs to every node. The major work of Data Reducer is to reduce the redundancy and merge information. The reduce rule is: if any of two alerts that destination IP and signature are respectively the same, the two would integrate as one and other attributes should be merged. For example, there are six alerts in metadata shown as table 1, and table 2 is the result. I1 and I2 are the same approach because that attacker does the method twice at different time. After analysis job worked, I1 and I2 reduced as R1. The different attribute, I3's b and I4's c, should be merged as R2's {b, c} because they have the same attributes, destination IP and signature. The example is that there are two hosts using the same method to attack one target. Any of the major attributes, signature and destination IP, are different, reduce job would not merge. For example, I5 and I6 are respectively on behalf of R3 and R4.

ID	**IP_Dst**	**Signature**	**Others**
I1	IP 1	X	{a}, {b},…
I2	IP 1	X	{a}, {b},…
I3	IP 2	X	{a}, {b},…
I4	IP 2	X	{a}, {c},…
I5	IP 3	X	{a}, {b},…
I6	IP 3	Y	{a}, {b},…

Table 1:- *Initial Log Data Sample*

ID	**IP_Dst**	**Signature**	**Others**
R1	IP 1	X	{a}, {b},…
R2	IP 2	X	{a}, {b,c},…
R3	IP 3	X	{a}, {b},…
R4	IP 3	Y	{a}, {b},…

Table 2:- *Log Data After MapReduce*

V. EXPERIMENTAL RESULTS

In order to measure performance of the proposed system, we constructed a cluster computing system composed of 5 systems of Pentium Core i5 processor. A master node that has a NameNode and a TaskTracker has 4 Gbytes RAM. Apache Hadoop 2.1.0 is installed on Ubuntu 13.04 operating system. Java development package JDK 1.7.0 is used. The nodes are connected by 100 Mbps Ethernet network switch. Block size in HDFS is set to 64Mbytes and the number of duplicate is set to 2.

In our first experiment we captured the SNORT IDS log report of simulated attack from distributed IDS sensors. The total size of the files is about 12 Gbytes at dIDS control center. We moved these data to HDFS for further processing.

Table 3 shows execution time when a single computer system without Hadoop is used, and when the proposed distributed Snort system with various number of slave nodes is used. Since there is no overhead on resource or task management in a single computer system, single system has better performance than the proposed system with a single slave node does. As the number of nodes increases in the proposed system, execution time decreases. From two nodes system, the performance of the proposed system starts to exceed that of the single system. When the number of nodes in the system is 5, performance of the system is about 2.53 times better than that of the single system. Fig. 6 shows that Gradient of speed up is almost linear, which means that the proposed system is highly scalable.

Number of Nodes	Execution Time (sec)	Speed Up Based on Hadoop
Single System	901.3	
1	1472.2	
2	656.9	224 %
3	601.7	244 %
4	465.3	316 %
5	355.5	414 %

Table 3:- *Comparison of Execution Time and Speed Up*

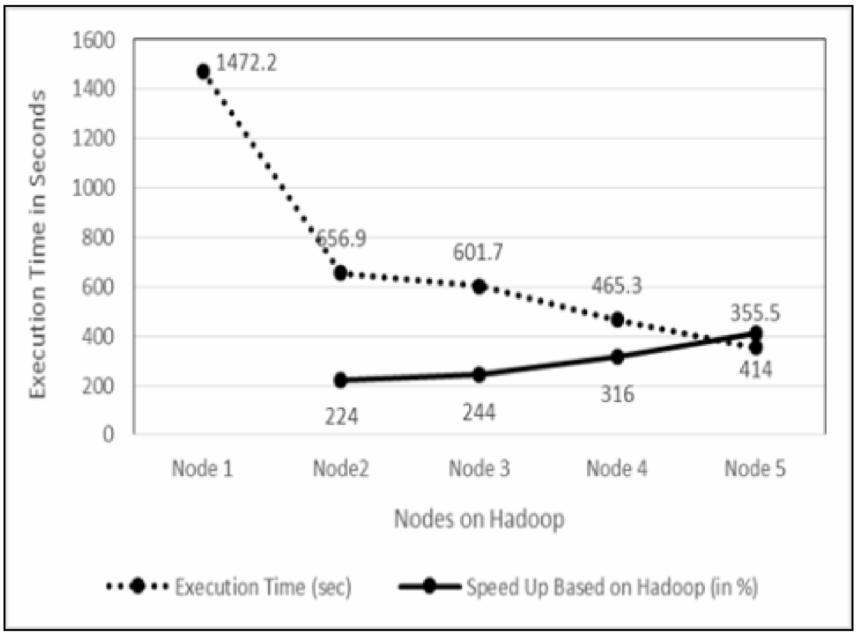

Fig 7:- *Speed up for the number of Nodes*

In our second experiment we did the analysis of dIDS log analysis execution time on dIDS Control Center and execution time analysis on Hadoop system of 5 nodes (Table4). We did the analysis of log size between 4 Gbytes to 12 Gbyte. The execution time analysis as shown in Fig 8 clearly indicate that the performance of Hadoop system is approx. 2.5 times better than the dIDS Control Center.

Size of IDS Log (in Gbytes)	Execution Time (in sec) on Hadoop (5-Nodes)	Execution Time (in sec) on dIDS Control Center
4	355.5	1056.5

5	391.05	1153.15
6	413.25	1209.75
7	448.35	1305.05
8	471.65	1364.95
9	501.65	1444.95
10	528.85	1516.55
11	550.85	1572.55
12	580.05	1650.15

Table 4:- *Hadoop Vs. Control Center dIDS Execution Time Comparison*

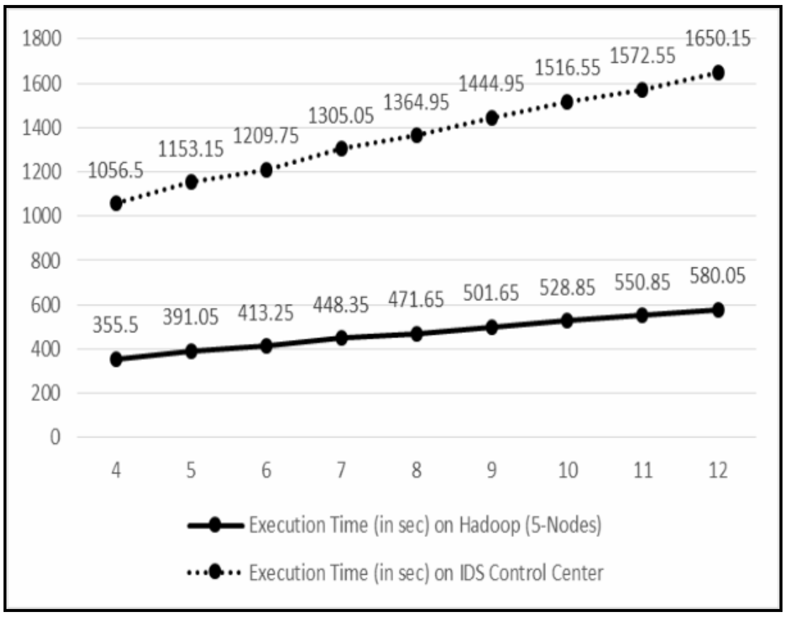

Fig 8:- *Hadoop Vs. Control Center dIDS Execution Time Performance Analysis*

We also simulated the operational performance of the IDS with KDD99 dataset categorized into five main classes (one normal class and four main intrusion classes: probe, DOS, U2R, and R2L).

1 Normal connections are generated by simulated daily user behavior such as downloading files, visiting web pages.

2 Denial of Service (DoS) attack causes the computing power or memory of a victim machine too busy or too full to handle legitimate requests. DoS attacks are classified based on the services that an attacker renders unavailable to legitimate users like apache2, land, mail bomb, back, etc.

3 Remote to User (R2L) is an attack that a remote user gains access of a local user/account by sending packets to a machine over a network communication, which include sendmail, and Xlock.

4 User to Root (U2R) is an attack that an intruder begins with the access of a normal user account and then becomes a root-user by exploiting various vulnerabilities of the system. Most common exploits of U2R attacks are regular buffer overflows, load-module, Fd-format, and Ffb-config.

5 Probing (Probe) is an attack that scans a network to gather information or find known vulnerabilities. An intruder with a map of machines and services that are available on a network can use the information to look for exploits.

The results of the simulation are shown in Table 5.

Activity	**Detection Rate (%)**	**False Positive (%)**
Normal	98.22	0.10
Probe	97.21	0.17
DOS	98.45	0.14
U2R	86.42	10.57
R2L	96.75	0.21

Table 5:- *Operational Performance of the IDS*

VI. CONCLUSION

In this paper we have explained the architectural design and performance analysis of a dIDS system based on Cloud Computing infrastructure. The experiments conducted with the cloud infrastructure intended as the central component of the proposed architecture shows that it is feasible to store and process massive quantities of data in a Hadoop environment. The significant benefits to build dIDS system on Cloud platform are its scalability and reliability. The experimental result was positive and we found that this work can be continued with several other improvement and performance analysis.

ACKNOWLADGEMENTS

I would like to thank MSRIT Management, my colleagues and Dept. of Computer Science and Applications, Bangalore University, for their valuable suggestion, constant support and encouragement.

References

1. A Mouinji, B L Charlier, D Zampunieris, N Habra, “Distributed Audit Trail Analysis”, Proceedings of the ISOC 95 Symposium on Network and Distributed System Security”, pp. 102- 112, 1995
2. ApacheTM Hadoop@ homepage, http://hadoop.apache.org/.
3. Axelsson, Stefan. Intrusion detection systems: A survey and taxonomy. Vol. 99. Technical report, 2000.
4. E H Spafford, D Zamboni, “Intrusion detection using autonomous agents”, Computer Networks, 34, pp. 547-570, 2000
5. H.Debar, M. Dacier, and A. Wespi, “Towards a taxonomy of Intrusion Detection Systems”, Computer Networks, vol 31, n0. 8, pp. 805-822, 1999.
6. Hadoop, http://hadoop.apache.org/.
7. Holtz, Marcelo D. ; Bernardo David ; Sousa Jr., R. T. . Building Scalable Distributed Intrusion Detection Systems Based on the MapReduce Framework. Telecomunicacoes (Santa Rita do Sapucai), v. 13, p. 22-31, 2011.
8. J. Dean and S. Ghemawat, MapReduce: Simplified Data Processing on Large Cluster, USENIX OSDI,2004.

ISSN 1512-1232

9. Jeong Jin Cheon and Tae-Young Choe, “Distributed Processing of Snort Alert Log using Hadoop”, IJET,Vol 5, No-3, Page 2685-2690, Jun-Jul 2013,
10. Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler, “The Hadoop Distributed File System,” IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp.1-10, 2010.
11. Nathan Einwechter,” An Introduction To Distributed Intrusion Detection Systems”, Security, Endpoint Protection (AntiVirus), SecurityFocus, 2001 (http://www.symantec.com/connect/articles/introduction-distributed-intrusion-detection-systems)
12. P A Porras, P G Neumann, “EMERALD: event monitoring enabling response to anomalous live disturbances”, Proceedings 20th National Information Security Conference, NIST 1997.
13. R A Kemmerer, “NSTAT: a Model-based Real-time Network Intrusion Detection System”, Technical Report TRCS97-18, Reliable Software Group, Department of Computer Science, University of California at Santa Barbara, 1997.
14. S R Snapp, J Bretano, G V Diaz, T L Goan, L T Heberlain, C Ho , K N Levitt, B Mukherjee, S E Smaha, T Grance, D M Teal, D Mansur, “DIDS (Distributed Intrusion Detection System) – motivation architecture and an early prototype”, Proceedings 14th National Computer Security Conference, Washington DC, October, pp. 167-176, 1999.
15. S Staniford-Chen, S Cheung, R Crawford, M Dilger, J Frank, J Hoagland, K Levitt, C Wee, R Yipi, D Z Erkle, “GriDS – a large scale intrusion detection system for large networks”, Proceedings 19th National Information Security Conference, Vol. 1, pp. 361-370, 1996.
16. S. Ghemawat, H. Gobioff, and S. Leung, The Google file system, ACM SOSP, 2003.
17. Yeonhee Lee and Youngseok Lee. 2012. Toward scalable internet traffic measurement and analysis with Hadoop. *SIGCOMM Comput. Commun. Rev*. 43, 1 (January 2012), 5-13. DOI=10.1145/2427036.2427038 http://doi.acm.org/10.1145/2427036.2427038

Article received: 2013-12-18