Eavesdropper: A Framework for Detecting the Location of the Processed Result in Hadoop

Chuntao Dong[1,2], Qingni Shen[1,2(✉)], Wenting Li[1,2], Yahui Yang[1,2], Zhonghai Wu[1,2], and Xiang Wan[1]

[1] School of Software and Microelectronics, Peking University, Beijing, China
{chuntaodong, wenlingli}@pku.edu.cn,
cowforkl990@hotmail.com,
{qingnishen, yhyang, wuzh}@ss.pku.edu.cn
[2] MoE Key Lab of Network and Software Assurance,
Peking University, Beijing, China

Abstract. Hadoop has become increasingly popular as it rapidly processes big data in parallel, while security mechanisms have been introduced or studied for Hadoop. In addition, other security issues that should not be neglected still exist. Data leakage is one of the major security challenges. This paper studies the vulnerability of authorization mechanism of services in Hadoop and the threat of information leakage. Some authorization mechanism allow all users to access services by default, an adversary can utilize these services to collect information of other users. We design and implement Eavesdropper, a framework which utilizes k-means clustering to address the nodes that store the processed results. We conduct a comprehensive of experiments, which clearly demonstrate that our detection framework is capable of detecting the nodes that store the results.

Keywords: Hadoop · MapReduce · YARN · Security · Data leakage · k-means

1 Introduction

Hadoop has become a major platform for big data. However, the research on Hadoop has mainly focused on the performance, and the security issues have not received sufficient attention. For Hadoop's initial purpose, it was always assumed that clusters in a trusted environment [9]. Therefore, since there were few security controls within Hadoop, many accidents and security incidents happened in such environments [8]. To enhance Hadoop system security, authentication and authorization are definitely necessary [2]. To our knowledge, insider attack can use the vulnerability of the security mechanisms configured by default to steal key value. We propose a framework for detecting the nodes that store the key value to prove the threat. Because the distributed characteristics of Hadoop, it is hard for insider attacker to locate the key value. To address these concerns, we propose Eavesdropper which is a novel, modular detection framework that can locate the key value.

S. Qing et al. (Eds.): ICICS 2015, LNCS 9543, pp. 458–466, 2016.
DOI: 10.1007/978-3-319-29814-6_39

Our Contribution. In summary, our work makes three key contributions:

- *The vulnerability analysis of the authorization mechanism in Hadoop:* If some authorization of services are configured by default is unsafe. We have highlighted the importance of ensuring the authorization of Hadoop is not configured by default.
- *A framework for detecting the nodes that store results:* If existing security mechanisms is not configured correctly, the adversary can get sensitive information of other users, but adversary cannot verify these nodes artificially. We propose a framework for detecting the DataNodes that store results.
- *Combination of vulnerability and detection framework:* Based on our detecting framework, we utilize the vulnerability of security mechanism that allows users to check information of applications in cluster and containers in a node to implement a detection scheme.

Paper Organization. We introduce the background in Sect. 2, and analyze threat and propose the detection framework in Sect. 3. The proposed detection scheme is elaborated in Sect. 4. Section 5 presents the implements and experimental results. We conclude the paper in Sect. 6.

2 Background

This section provides the background information on YARN and security mechanism of Hadoop [3–5].

YARN. In Hadoop 2.0 the classic MapReduce [7] module is upgraded into a new computing platform, called YARN [12]. YARN provides the ability to execute user code across machines in a cluster. This user code is executed in the container. Each container has an identity (Container ID). An application instance also has an identity (Application ID). There is a relation between them. Container ID is achieved by using the application ID along with a monotonically increasing counter for the container. This paper aims to address the vulnerability of unconstrained web port in Hadoop and the attacks of stealing high-value information to a Hadoop cluster with multi-tenancy. YARN provides system management interface for user to browse information of applications running in the cluster and containers running in the nodes. The default HTTP port is 8088, it is deployed on RM showing the Applications' information, such as the current queue backlog, resource utilization, application execution and so on, as illustrated in Fig. 1. This web UI opens up result in a number of potential issues [1].

The Security Mechanism of Hadoop. System security mechanism usually consists of two parts: authentication and authorization. Hadoop security relies on Kerberos and Tokens for authentication and relies on access control list (ACL) for authorization [10]. In Hadoop, Kerberos is used for client authentication at the "entry points" only. These entry points are master services like the NameNode, RM and HistoryServer. By default, the Kerberos authentication is disabled for Hadoop, which default is "simple".

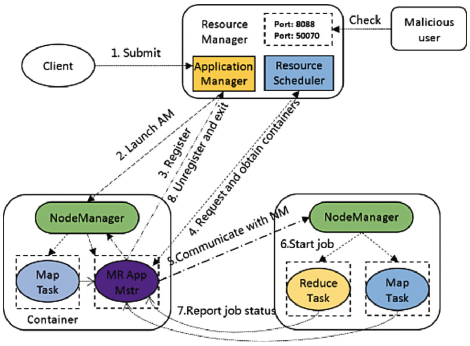

Fig. 1. The architecture of YARN

Hadoop relies on access control list (ACL) for authorization. According to the authorized entity, it can be divided into Map Reduce job Queue ACLs, Map Reduce Job ACLs, Service Endpoint ACLs [8]. Service Endpoint ACLs: All RPC endpoints can have ACLs applied at the protocol layer. These ACLs can control the users and groups that can access a given service protocol.

3 Vulnerability and Threat

In this section, we analyze the vulnerability of security mechanism in Hadoop and present a framework for detecting the DataNodes that store key value.

Vulnerability Analysis. As discussed in Sect. 2, Hadoop employ authentication and authorization to enhance system security. Hadoop relies on access control list (ACL) for authorization. These ACLs can control the users and groups that can access a given service protocol. By default, some protocols open to all users and groups. We believe that any shared web port left unconstrained will be a vulnerability. This vulnerability may be utilized by malicious to threaten system security and steal data of other users in the cluster. In the next part of this section, we propose a detection framework for detecting the DataNodes that store results to prove the threat of leaking key value pairs.

Threat Model. We assume adversary is regarded as trusted but have malicious intentions. He tries to steal sensitive results from other user that he is not allowed to access. The model of the detecting the nodes that store the results is depicted in Fig. 2. There are three types of entities:

User: These entities have data to be stored in the cluster and interact with the Hadoop Cluster to manage their data and submit applications on the cluster.

Adversary: The adversary intends to steal the processed results of other users in the cluster. He collects information by utilizing the service of cluster and analyzes the nodes that store results.

Hadoop Cluster: The Hadoop cluster provides resources and services for users.

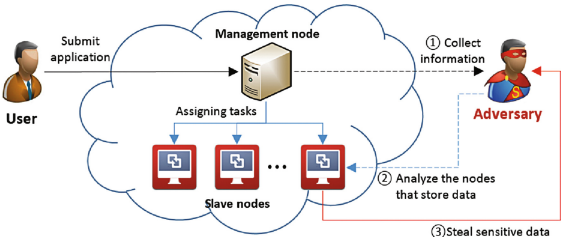

Fig. 2. The model of detecting the node storing the results

In the following section, we proposed a detection framework based on the threat model.

4 Detection Framework

We start with a sample scenario where the cluster is only shared between two parties, i.e., the adversary and the victim. Based on the threat model, we proposed Eavesdropper to locate the nodes that store results. Figure 3 demonstrates a sample architecture of Eavesdropper. The location detection mainly consists of two stages: Probing and Analysis. In our analysis for above identified threat, we have the following assumption.

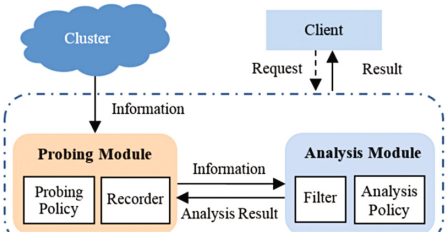

Fig. 3. A sample architecture of Eavesdropper.

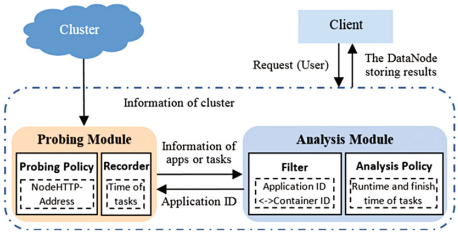

Fig. 4. A detailed architecture of Eavesdropper.

Assumption: The security mechanism is configured by default, users can easily access the services and get information of tasks in the cluster.

Probing. If an adversary and a victim are running synchronously, the adversary can trace the running process of victim's application and collect information about the application. For example, the adversary can utilize open service port to get information of the application. We should implement the probing policy according to practical scene. After capturing the information, we use the recorder to record the information.

Analysis. An adversary can utilize the information about the target application to analyze the location of results. Before analyzing the information, we need to use the filter to screen the information that is related to the target application. Such as, we can use the runtime and the finish time of a task to analyze the type of the task.

4.1 Detailed Eavesdropper

In the previous analysis, we describe our main ideas for addressing the two stages respectively. Note that we focus on the probing and analysis, and discuss the detailed design of the detection scheme in the next two part of the Section. We extend Eavesdropper by utilizing the open service port and related mechanism of YARN. Figure 4 demonstrates a detailed architecture of Eavesdropper. We realize Eavesdropper by utilizing the open service port and the relation between Application ID and Container ID. In the following of this section, we will describe our scheme in detail. We examine Eavesdropper in Sect. 5.

4.2 The Design of Probing Module

In this part, we will introduce the implement of probing module. The main function of probing module is that collect information of applications and tasks. The adversary intends to steal processed result of target user. He needs to collect and analyze information to confirm which application is submitted by the target user firstly. The adversary can achieve this by scanning the management interface of all applications. The interface including the information of applications, such as Application ID, user name and progress etc. The recorder record the related information of target user and send information to the analysis module. Using these information, the analysis module can get the Application ID of the application submitted by target user.

After receiving the application ID, the probing module will collect information of the application. An application needs to apply for a mass of containers for computing. Each node has a mass of containers, we must identify which container belongs to the target application. In the Sect. 2, we have known the relation between Container ID and Application ID. We will use this relation to collect information of target application. The process of collecting and recording information of the target application is cyclic until the application finish running. The steps are detailed below:

(1) Scan nodes management interface of the cluster to get the NodeHTTPAddress of all the nodes in the cluster. The address is the address to access each node of the cluster.
(2) Access the NodeHTTPAddress of each node to collect and record the information of containers that belong to the target application on the node.
(3) Using the relation between Container ID and Application ID to screen out the containers that belong to the target application.
(4) Using the recorder to record the container IDs, running node, start time and finish time of each container. We will use the record of containers to analyze the type of the task running in the container in the subsequent section.

4.3 The Design of Analysis Module

The analysis module is the critical module in proposed detection scheme and its main function is to analyze the node that store result of the target application. In the last part, we have recorded the information of containers. We need to analyze which container was used for a reduce task. Because we know that the result of a mapreduce job is generated by reduce task. Compared to map task, the reduce task finish later because the reduce task need to wait the relevant map task finish running. When the reduce task finish running, the node that the reduce task running on will upload the result to the HDFS at once [11]. HDFS's placement policy is to put one replication on the local node firstly, we can confirm that one replication of the result is on the local node.

The Analysis Policy. Our goal is to find out all the reduce task(s) of the target job. We provide a solution that utilizes k-means clustering [6]. In our solution, we utilize the runtime and the finish time of tasks to analyze the type of tasks. The runtime of reduce tasks is longer than that of map tasks and the reduce tasks finish later than map tasks. Our solution aims to partition the n observations into 2 sets $S = \{S_{map}, S_{reduce}\}$, where each observation is a 2-dimensional real vector that consists of the runtime R_n and finish time T_n of each tasks. The computing task that firstly start to run must be a map task, and the computing task that finish running last must be a reduce task. We initial (T1, R1) to μ_1 and (T_n, R_n) to μ_2. The algorithm proceeds by alternating between two steps:

Assignment step: Assign each observation to the cluster whose mean yields the least within-cluster sum of squares. Our goal is make the value of J that in the formula (1) as small as possible, if data point n is categorized into cluster k, r_{nk} equals 1, otherwise r_{nk} equals 0.

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2 \tag{1}$$

$$\mu_k = \frac{1}{N_k} \sum_{j \in clusterk} x_j \tag{2}$$

Update step: Calculate the new means to be the centroids of the observations in the new clusters. The value of μ_k is the average value of all the data points of cluster k. We use the formula (2) to calculate the new center point of each cluster.

After we have detected these DataNodes that store result, we need to find the result in the DataNode. We assume that we have invaded in these DataNodes. It is difficult to pick out the datablock of the target application in a DataNode. We utilize a relation between the finish time of reduce task and the modified time of data block to search the result. The modified time approximatively equals the finish time of reduce task plus the time that upload data. We can seek out the data block based on the relation.

The problem of the solution is that the failed tasks may be clustered by mistake. For example, when a map task may be delayed, the map task will run longer and finish later that may make it looks like a reduce job. The data that we aim to steal may be encrypted after computed, but encrypt do not mean safe completely. Our main focus is

the process of computing data, data is unencrypted in the computing process, we can steal data before data encrypted and uploaded to HDFS.

5 Implementation and Experimental Results

5.1 Implementation

We have implemented a detection software in JAVA language. We conducted several experiments using the local 64-bit Centos operation system with an Intel Core i7 processor running at 3.4 GHz, 4096 MB of RAM, and run Hadoop 2.6.0. The configuration of our Hadoop cluster is of one NameNode, one RM, ten NodeManagers, ten NataNodes.

5.2 Experiment Results

In this section, we select a standard benchmark for evaluating our detection solution. The 7 benchmark applications cover a wide range of data-intensive tasks: compute intensive, shuffle intensive, database queries, and iterative. The size of the input data is between 1 GB and 1.5 GB in these case studies. We run 20 experiments of each benchmark applications to verify if our scheme can detect the reduce tasks accurately and efficiently.

In the first experiment, we run a wordcount job that consists 6 map tasks and 2 reduce tasks to verify the availability of our solution. We record the finish time and runtime of all the tasks and analyze the record. According to the analysis policy, we choose two initial means Container_03(C_03), Container_08 as μ_1, μ_2. Then, we use the k-means clustering to partition the n computing tasks into 2 sets $S = \{S_{map}, S_{reduce}\}$. The partition result is S_{map} = {C_02, C_03, C_04, C_05, C_06, C_07} and S_{reduce} = {C_08, C_09}. We run several jobs including target job and several other jobs in the other runs simultaneously. Our solution also detected the reduce tasks successfully and efficiently (Table 1).

Table 1. A wordcount job that consists of 6 map tasks and 2 reduce tasks.

Container ID	Start time	Finish time	Runtime(s)	Task type	Analysis result
Container_01	17:10:20	17:11:01	41	AplicationMaster	
Container_02	17:10:28	17:10:44	16	Map task	
Container_03	17:10:26	17:10:43	17	Map task	
Container_04	17:10:28	17:10:45	17	Map task	
Container_05	17:10:27	17:10:43	16	Map task	
Container_06	17:10:28	17:10:46	18	Map task	
Container_07	17:10:30	17:10:45	15	Map task	
Container_08	17:10:47	17:10:56	9	Reduce task	Rack1Node5
Container_09	17:10:48	17:10:55	7	Reduce task	Rack1Node3

We use other 6 benchmark applications to evaluate the accuracy of our solution. We summarize and analyze the accuracy of 7 benchmark applications, as shown in Table 2. We find that the analysis accuracy of some benchmark application is not 100 %. By analyzing the feature of each type of benchmark applications, we find out the applications which the runtime of map and reduce tasks is similar are easily interferential. We will improve our solution in the future work and make our detection software more reliable.

Table 2. Summary of the runtime of tasks and the analysis accuracy of 7 benchmark applications.

Job type	Num. of experiments	Num. of map tasks	Num. of reduce tasks	Runtime of map tasks	Runtime of reduce tasks	Num. of analysis successfully	Analysis accuracy
Wordcount	20	6	2	17.2	7.8	20	100 %
Index	20	8	5	50.5	15.7	19	100 %
Grep	20	8	5	18.9	15.4	16	80 %
Aggregate	20	16	8	4.7	10.1	18	90 %
Join	20	16	8	7.8	22.8	19	95 %
Pagerank	20	24	10	10.8	24.6	19	95 %
Kmeans	20	24	10	18.2	37.5	18	90 %

6 Conclusion

In this paper, we proposed a framework for detecting the node that stores key value and analyzed the vulnerability of the authorization mechanism of services in Hadoop. Combining these two aspects, we implemented a new detection scheme to detect the nodes that store the processed result in MapReduce successfully. Our experiment results demonstrated the effectiveness of our detecting framework of three cases in real-world systems, we had confirmed that the proposed detection program based on detection framework can detect the nodes that store the processed result.

Although our framework and detection scheme is implemented in experimental environment, we can improve and use it in the cloud with multiple users. In the future work, we aim at extending the approach to a larger set of application level vulnerabilities and propose a learning algorithm to classify type of tasks, as well as defining a sophisticated method able to detect Information Eavesdropping attacks in the cloud computing environment.

Acknowledgment. This work is supported by the National High Technology Research and Development Program ("863" Program) of China under Grant No. 2015AA016009, the National Natural Science Foundation of China under Grant No. 61232005, and the Science and Technology Program of Shen Zhen, China under Grant No. JSGG2014051 6162852628.

References

1. Apache hadoop. http://hadoop.apache.org
2. Sharma, A., Kalbarczyk, Z., Barlow, J.: Analysis of security data from a large computing organization. In: IEEE 41st International Conference on Dependable Systems Networks, pp. 506–517. IEEE (2011)
3. Hadoop in Secure Mode. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SecureMode.html
4. Ulusoy, H., Colombo, P., Ferrari, E.: GuardMR: fine-grained security policy enforcement for mapreduce system. In: ASIACCS 2015, pp. 285–296. ACM (2015)
5. Lahmer, I., Zhang, N.: MapReduce: MR model abstraction for future security study. In: International Conference on Security of Information and Networks, pp. 392–398. ACM (2014)
6. Hartigan, J.A., Wong, M.A.: Algorithm AS 136: a k-means clustering algorithm. J. Roy. Stat. Soc. Ser. C **28**(1), 100–108 (1979). Wiley for the Royal Statistical Society
7. Dean, J., Ghenmawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI 2004, pp.137–150. ACM (2004)
8. Huang, J., Nicol, D.M., Campbell, R.H.: Denial-of-Service threat to hadoop/YARN clusters with multi-tenancy. In: IEEE International Congress on Big Data (2014)
9. Smith, K.T.: Big Data Security: The Evolution of Hadoop's Security Model (2013)
10. O'Malley, O., Zhang, K., Radia, S.: Hadoop security design. In: Yahoo! Tech Rep (2009)
11. White, T.: Hadoop: The Definitve Guide, 3rd Edition, pp. 43–79. O'Reilly, Sebastopol (2012)
12. Vavilapalli, V.K., Murthy, A.C., Douglas, C.: Apache hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing, vol. 5. ACM (2013)