2015 6th International Conference on Information and Communication Systems (ICICS)

Enhancing Security of Hadoop in a Public Cloud

Xianqing Yu, Peng Ning, Mladen A. Vouk
Computer Science Department
North Carolina State University, Raleigh, North Carolina, USA
Email: {xyu6, pning, vouk}@ncsu.edu

***Abstract*—Hadoop has become increasingly popular as it rapidly processes data in parallel. Cloud computing gives reliability, flexibility, scalability, elasticity and cost saving to cloud users. Deploying Hadoop in cloud can benefit Hadoop users. Our evaluation exhibits that various internal cloud attacks can bypass current Hadoop security mechanisms, and compromised Hadoop components can be used to threaten overall Hadoop. It is urgent to improve compromise resilience, Hadoop can maintain a relative high security level when parts of Hadoop are compromised. Hadoop has two vulnerabilities that can dramatically impact its compromise resilience. The vulnerabilities are the overloaded authentication key, and the lack of fine-grained access control at the data access level. We developed a security enhancement for a public cloud-based Hadoop, named SEHadoop, to improve the compromise resilience through enhancing isolation among Hadoop components and enforcing least access privilege for Hadoop processes. We have implemented the SEHadoop model, and demonstrated that SEHadoop fixes the above vulnerabilities with minimal or no run-time overhead, and effectively resists related attacks.**

Keywords*—Public cloud; security; compromise resilience; least access privilege; lack of fine-grained access control; overloaded authentication key.

I. Introduction

Map-Reduce is a framework often used today in the context of big data. Hadoop is an implementation of the Google Map-Reduce framework. Many companies, such as Wal-Mart (WMT), Walt Disney (DIS), Bank of America (BAC), etc., use Hadoop to analyze very large data sets. A large Hadoop cluster can have more than 42,000 compute nodes [1]. Cloud computing is an emerging technology which gives reliability, flexibility, scalability, elasticity and cost saving to cloud users. There have been several successful efforts to move Hadoop into cloud [2], [3]. Amazon Elastic MapReduce (Amazon EMR), Cloud::hadoop [3] and Joyent solution for Hadoop [2] are commercial cloud platforms.

However Hadoop was originally designed to run in a well controlled private environment. When moving Hadoop to a public cloud, there are challenges to original Hadoop security mechanisms. In a concurrent resource sharing cloud, hardware resources (e.g. CPU, memory, hard disks, network card) and software resources, (e.g. hypervisor, operating system) can be shared among cloud users [4]. In such a resource sharing environment, security often relies on the high-level of isolation of users' workloads [5], [6]. This isolation is often enforced through virtualization technology. A hypervisor or an OS is the highest privilege software to manage all hardware resources, and to enforce the isolation. A hypervisor itself can be a target of attacking. Hypervisor vulnerabilities have been reported in the past few years [7], [8], [9], [10]. In the work [11], F. Rocha demonstrated how a malicious hypervisor can easily obtain passwords, cryptographic keys, files and other confidential data. In [12], S. Bugiel et al. displayed various unsafe publicly available and widely used Amazon Machine Images (AMIs). T. Ristenpart et al. showed how to determine if two instances were co-resident on the same physical machine, and launched side-channel attack in Amazon EC2 [13]. All these internal cloud attacks can bypass current Hadoop security mechanisms.

There is a rising concern that Hadoop in its present form may not be able to maintain the same security level in a public cloud as it does in a protected environment. These internal cloud attacks can bypass Hadoop security mechanisms to compromise the safety of data and computing in Hadoop. Many works are being done to improve overall cloud security [4], [14], [15]. We work on the security of Hadoop from the other perspective, that is to improve compromise resilience of Hadoop by redesigning Hadoop security mechanisms. When some Hadoop components are compromised, the security level of Hadoop degrades gracefully instead of vanishing.

Most of cloud attacks mentioned above aim for infrastructure as a service (IaaS) which is widely adopted. In this paper, we evaluated how Hadoop resists internal cloud attacks in a public cloud based on IaaS model. We developed and implemented a new SEHadoop model, which includes a SEHadoop runtime model, SEHadoop Block Token and SEHadoop Delegation Token, to improve compromise resilience of Hadoop through two major aspects: enhancing isolation level among Hadoop components and enforcing least privilege access control on Hadoop processes. High isolation level among Hadoop components can prevent compromised Hadoop processes from compromising the rest parts of Hadoop. Least privilege access control can limit the range of data a compromised Hadoop process can access.

The rest of this paper is organized as follows. In Section 2, we discuss the threats in a public cloud. In Section 3, we present SEHadoop model. In Section 4, we describe the details of SEHadoop implementation and experiments. In Section 5, we review related work. In Section 6, we conclude this paper.

II. The threats in a public cloud

In a public cloud, multiple tenants share hardware and software resources. In IaaS model, if attackers can ensure their VMs to share the same resources with Hadoop's VMs in a public cloud, they can launch various internal cloud attacks.

First, the threats can come from compromised hypervisors. Attackers can exploit vulnerabilities in hypervisors [7], [8], [9], [10], and use methods demonstrated by F. Rocha et al. [11] to obtain secret keys in Hadoop VM's file system and memory.

978-1-4799-7349-1/15/$31.00 ©2015 IEEE

Second, the attacks can be launched from malicious VMs. In the work of T. Ristenpart et al. [13], they showed how to determine if two instances are running on the same physical machine, and launch side-channel attack in Amazon EC2.

Third, attackers can leverage the mistakes made by Hadoop administrators to start attacks. As demonstrated by S. Bugiel et al. [12], many users forget to delete secret keys in images before publishing their images in public. Hadoop administrators may forget to delete the secret keys which are used by Node Managers and Data Nodes to authenticate themselves from Kerberos before publishing Hadoop images. Hadoop uses Kerberos for initial authentication during initialization, and each Data Node and Node Manager uses its own secret key to authenticate itself. Attackers can use these keys to impersonate legitimate parts of Hadoop.

In a nutshell, while Hadoop running on a public cloud, attackers can launch internal cloud attacks to bypass current Hadoop security mechanisms, and steal sensitive information in a large scale Hadoop. One critical problem is that when one or some Hadoop components being compromised, how much security level in Hadoop remains. We evaluated security mechanisms in current Hadoop, and identified two major vulnerabilities in two widely used authentication/authorization token, a Block Token and a Delegation Token.

A Block Token is used by a process to access data on a Data Node. During HDFS initialization, a Name Node randomly generates a symmetric key and distributes it to Data Nodes when Data Nodes are booting up and contacting the Name Node for registration for the first time. The key is used to generate Block Token Authenticator (BTA) and Block Token. A Name Node and all Data Nodes **share the same key** even in a large scale HDFS. If the key is leaked from machines of HDFS by any attack methods, an attacker has the capability to use the key to generate arbitrary Block Tokens as he or she wants to, and accesses any data blocks in HDFS. The overloaded authentication key in Block Token dramatically weaken the isolation among components of HDFS.

Delegation Token is used by map and reduce processes to authenticate themselves through a Name Node when they access HDFS. **The vulnerability is that Delegation Token lacks fine-grained access control.** With a Delegation Token, a process has the privilege to behave as the Hadoop user and to access all content which the Hadoop user is allowed to access. Loss of a non-expired Delegation Token can potentially enable the attacker to impersonate the Hadoop user and access the lost Delegation Token's owner's data. A Delegation Token is used by all computing processes (i.e. map processes and reduce processes which are represented as Containers in Hadoop). These Containers are running across Hadoop system. When Hadoop is running in a public cloud, Containers can also be the target of internal cloud attacks, and a leaked Delegation Token can be used steal a large amount of data from HDFS. Therefore, fine-grained access control must be enforced on Hadoop computing processes to improve compromise resilience.

III. SEHADOOP MODEL

A. Assumptions and Attack Models

The goal of the research is to improve Hadoop compromise resilience. When compromise happened in parts of Hadoop, the security level of Hadoop degrade gracefully, instead of losing all security protection. We assume that most of Hadoop components are running in a public cloud on IaaS model, and internal cloud attacks are possible. The attacks can be launched by using malicious hypervisors, co-reside VMs, mistakes of Hadoop administrators, and etc. A secure zone can be created to shield critical parts of Hadoop from internal cloud attacks, and administrators configure these critical Hadoop components correctly. The secure cloud zone has dedicated hardware and software resources which have a strong isolation from other public cloud users. Amazon EC2 has dedicated instance service. Other well controlled private systems can also be used for a secure zone. Since these services usually incur higher cost than a public cloud does, we need to control the scale of components running in a secure zone. We assume that attackers are only able to initiate the limited scale of internal cloud attacks on Hadoop components in a public cloud. In addition to security goal, we want to maintain low operation overhead in Hadoop, and incur no extra storage overhead in HDFS.

B. Overview of SEHadoop

To improve Hadoop compromise resilience, we design a new SEHadoop model through two major principles: enhancing isolation level among Hadoop components and giving least access privilege for Hadoop processes. When compromise starts in Hadoop, strong isolation level can help Hadoop limit the extent of compromise. It will enforce hackers to attack one component at a time and slow down the pace of attacks. Enforcing least access privilege for Hadoop components can ensure that compromised Hadoop processes can only access limited data. Comparing with original Hadoop, attackers need to attack more components in SEHadoop to steal the same amount of data. Since some Hadoop components may be running on a large number of VMs, such as Data Node, Node Manager and Container, they have larger possibility being attacked than other Hadoop components have, e.g a malicious VM of an attacker has better chance to co-reside with VMs running Data Node to launch internal cloud attacks comparing with VM running Name Node. We carefully examined these components to ensure that the security mechanisms they are using satisfy the two principles. The SEHadoop model consists of SEHadoop runtime model, SEHadoop Block Token and SEHadoop Delegation Token.

C. Enhance Isolation Level

Hadoop uses many kinds of secret keys and tokens to perform authentication and authorization among different Hadoop components. These sensitive keys and tokens may be used to attack Hadoop once leaked. In order to enhance isolation level, we want to maintain the level of keys and tokens sharing in Hadoop components to minimum, thus leaked keys and tokens in one Hadoop process can not be used to attack the rest of parts of Hadoop. Block Token has a shared key in entire HDFS, and one Delegation Token is shared by all processes of one job. It is necessary to improve security level of both tokens.

D. Least Access Privilege

Hadoop YARN processes read data from HDFS for computing. When YARN processes, such as Container, access data, only least access privilege should be given to. In current

Hadoop design, each Container can access entire data set of a user. In SEHadoop Delegation Token, we bring fine-grain access control to Containers and Node Managers, and each Container can only access the data it allowed.

E. SEHadoop Runtime Model

Some processes in Hadoop contain critical information. A Name Node manages the file system name-space and has the keys to generate Block Tokens and Delegation Tokens. An attacker can fetch all keys from its memory and access data from all active Data Nodes. A Resource Manager distributes all Delegation Tokens to proper Node Managers and Containers. These Delegation Tokens can be used by the hacker to access data of the Delegation Tokens' owners. An Application Master distributes the Delegation Token of one job to all the job's map and reduce processes. Once the keys or Delegation Tokens have been intercepted, an attacker can access a broad range of data in HDFS. A user uses a Job Client as interface to access HDFS and YARN, the Job Client contains sensitive information (e.g. Kerberos' ticket and Delegation Tokens) and is able to access all the user's data. A Node Manager which manages Application Masters is responsible for setting up proper configuration, booting up the Application Master and transferring the Delegation Token to the Application Master. An attacker can use it to intercept all Delegation Tokens sent to an Application Master. Hadoop uses Kerberos to conduct for most of authentication operations. Compromising of Kerberos, an attacker can impersonate any users in Hadoop. Therefore, a Name Node, a Resource Manager, Application Masters, Job Clients, Kerberos, and Node Managers which manages Application Masters should run in a secure zone. The rest of processes of SEHadoop, such as Data Nodes, Node Managers and Containers, can run in a public cloud.

Fortunately the scale of these processes is usually small or fixed. We propose a runtime model to protect these processes in a secure environment, named a secure zone, while other processes run in a public cloud environment. This is illustrated in the Figure 1. In a secure zone, a computer has dedicated resources to keep a public cloud threats away. In our model, we improve the overall Hadoop security level, while keeping most of Hadoop processes still in a public cloud environment. We minimize the number of SEHadoop processes running in a secure zone and the resource demands of SEHadoop in a secure zone. Storage processes (i.e. Data Nodes) and computing processes (i.e. Containers for map or reduce processes) usually have a great demand on resources, such as storage space, CPU, memory, etc., are all in a public cloud. In this model, we keep the need for a secure zone to the minimum, therefore SEHadoop can maintain the benefits of a public cloud, and protect some critical Hadoop processes.

F. SEHadoop Block Token

We designed SEHadoop Block Token to fix the overloaded authentication key vulnerability. SEHadoop Block Token uses secret keys to conduct for an authentication, but these keys are different for each Data Node. SEHadoop Block Token uses AES algorithm to encrypt plain-text of the token's content, thus SEHadoop Block Token's content remains protected during transfer from a Name Node to a Data Node. The SEHadoop Block Token's generation formats are as follows:

$$\texttt{SEBTID} = \{ \text{ED, KeyId, UId, BPId, BId, AM, CIP}\} \quad (1)$$

$$\texttt{SEBT} = \text{AES}(\text{ SEBTID, Hash}(\text{ SEBTID})) \quad (2)$$

Where ED stands for expiration date of Block Token, KeyId is used to identify which key is used, UID stands for User ID, BPId represents block pool ID, BId defines which block needs to be operated on, AM means Access Mode (e.g. read, write, replace, copy) for this authorized operation, CIP is HDFS Client's IP. SEBTID stands for SEHadoop Block Token ID which defines all necessary information for an HDFS access request, Hash is the Secure Hash Algorithm (SHA256) hash function which is used to generate hash value of SEBTID, AES represents Advanced Encryption Standard (AES) encryption function which is used to encrypt the plaintext of a Block Token, and SEBT represents SEHadoop Block Token.

In SEHadoop model, a Name Node uses different keys to generate an SEHadoop Block Token for each Data Node. A Name Node creates a key set for all Data Nodes, and manages the mappings from a key to a Data Node. Each Data Node has a unique symmetric key which is created by and shared with the Name Node. The keys are created when Data Nodes boot up and register with the Name Node. The symmetric keys are used in AES function in format (2) to generate SEHadoop Block Tokens.

In a Block Token of Hadoop, Block Token ID is transferred in plaintext from an HDFS Client to a Data Node. If such a Block Token ID is intercepted, its content become known to an attacker, and from this the attacker can deduce the topology of data in HDFS. SEHadoop Block Token uses AES and SHA256, which have low overhead, in format (2) to ensure SEHadoop Block Token ID's integrity and confidentiality.

A Data Node verifies an SEHadoop Block Token in two steps. First it decrypts the SEHadoop Block Token using its secret key shared with the Name Node. The Data Node gets the SEHadoop Block Token ID and the hash value from the decrypted SEHadoop Block Token. Second, the Data Node computes hash value of the SEHadoop Block Token ID and compares the result to the hash value from the decrypted SEHadoop Block Token. If both hash values match, this SEHadoop Block Token is verified. Then SEHadoop Block Token ID is used to authorize the access request.

G. SEHadoop Delegation Token

We designed SEHadoop Delegation Token to fix the lack of fine-grained access control vulnerability. SEHadoop Delegation Token must include fine-grained access control to allow a Name Node to restrict the content that the process can access when a Delegation Token is used. A Name Node creates a Delegation Token but does not know the fine-grained access control information. Therefore we propose to create SEHadoop Delegation Token in two steps. First, SEHadoop Delegation Token is created by a Name Node without fine-grained access control information. It carries data access authorization information and can be used then to generate fine-grained access control SEHadoop Delegation Tokens in the next step. Second, a process, which has fine-grained access control information (e.g. Job Client), uses the SEHadoop Delegation Token to create multiple Delegation Tokens and adds fine-grained access control information into them. At the end, the Name Node must be able to verify all SEHadoop Delegation Tokens regardless

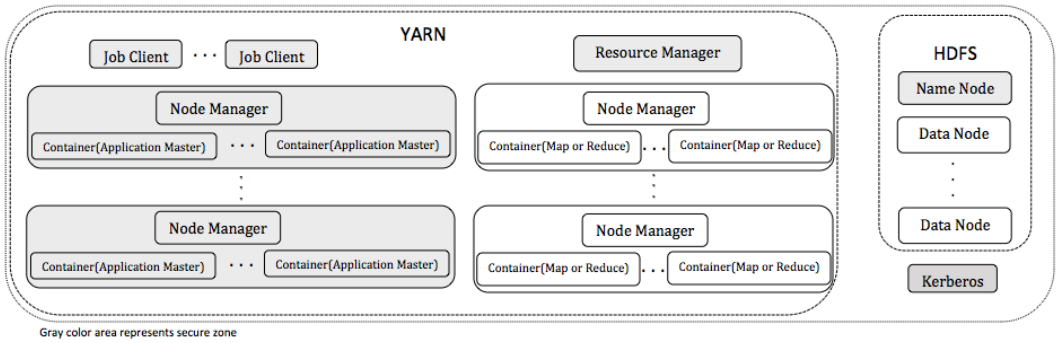

Fig. 1. SEHadoop Runtime Model. Job Clients, Node Managers and Containers running Application Masters, Resource Manager, Name Node, and Kerberos are in secure zone.

of whether they are generated by the Name Node in the first step or by other processes in the second step.

SEHadoop Delegation Token consists of two parts: Parent Delegation Token and Child Delegation Token. There is one Parent Delegation Token and multiple Child Delegation Tokens for each job. A Parent Delegation Token is created by a Name Node. A Parent Delegation Token is used by a Job Client to generate multiple Child Delegation Tokens. Parent Delegation Token's formats is described as follows:

$$\texttt{PDTokenID} = \{\text{ OI, RI, IssueDate, MD, SN}\} \quad (3)$$
$$\texttt{PDTA} = \text{ HMAC-SHA2(Key, PDTokenID)} \quad (4)$$
$$\texttt{PDToken} = \{\text{ PDTokenID, PDTA, CKey}\} \quad (5)$$

Where OI stands for the Parent Delegation Token's owner's ID, RI means renewer's ID, IssueDate shows the date this Parent Delegation Token is created, MD represents the maximum date which is effective time period of the Parent Delegation Token, SN is a sequence number for the Parent Delegation Token, PDTokenID is Parent Delegation Token ID, HMAC-SHA2 represents a keyed-hash message authentication code which the hash algorithm used is SHA-256, PDTA stands for Parent Delegation Token authenticator, CKey stands for Child Key, PDToken is Parent Delegation Token.

Comparing to Hadoop Delegation Token, a Parent Delegation Token has a Child Key for generating Child Delegation Tokens in a Job Client later. The Name Node stores the Child Key while creating the Parent Delegation Token and uses the key to verify Child Delegation Token later. The proposed Child Delegation Token's formats are following:

$$\texttt{SplitInfo} = \{\text{ Path, StartOffset, Length}\} \quad (6)$$
$$\texttt{CDTokenID} = \{\text{ PDTokenID, SplitInfo}\} \quad (7)$$
$$\texttt{ChildDTA} = \text{HMAC-SHA2(CKey, CDTokenID)} \quad (8)$$
$$\texttt{CDelegationToken} = \{\text{ CDTokenID, ChildDTA}\} \quad (9)$$

Where Path is the input file path, StartOffset defines the offset of first input byte in file, Length is data input's length, CDTokenID stands for Child Delegation Token ID, PDTokenID means Parent Delegation Token ID, CKey represents Child Key, ChildDTA stands for Child Delegation Token authenticator, HMAC-SHA2 represents a keyed-hash message authentication code which the hash used is SHA-256, CDelegationToken stands for Child Delegation Token.

Child Delegation Tokens are created by a Job Client. A Job Client initializes a job and splits the input data for every map process. SplitInfo in format (6) defines the specific access control information, such as a file's path, first byte of offset in the file, and input data's length. A Parent Delegation Token ID and a SplitInfo comprise a Child Delegation Token ID in format (7). A Child Key is created by a Name Node and included in a Parent Delegation Token, so it can be used to generate a Child Delegation Token Authenticator in function (8). A Job Client can use a Child Key to generate Child Delegation Tokens without number limitation, and each map process can have its specific Child Delegation Token. These Child Delegation Tokens will be passed to a Resource Manager in a job submission and transferred to each map or reduce process. When a process uses its specific Child Delegation Token to access HDFS, the Child Token ID is sent to a Name Node. The Name Node uses the Parent Token ID which is inside the Child Token ID to search for which Parent Delegation Token this Child Token ID belongs to. The Name Node uses the Parent Delegation Token's Child Key to compute a new Child Delegation Token Authenticator based on Child Token ID received from the process. If the process has a correct Child Delegation Token, it must have the same Child Delegation Token Authenticator as the Name Node generated. Then the process and the Name Node have the same Child Delegation Token Authenticator as the secret and use SASL/DIGEST-MD5 protocol to finish authentication.

H. Security Analysis

In order to maintain a strong isolation level among components of Hadoop in a public cloud, Hadoop components should avoid sharing secret keys or tokens. Therefore when one Hadoop component is compromised, its leaked keys and tokens will not expose keys and tokens of other Hadoop components. In SEHadoop, SEHadoop Block Token fixes the overloaded authentication key issue and resists attackers targeting storage processes (e.g. Data Nodes). Each Data Node is forced to share a unique symmetric key with the Name Node for generating SEHadoop Block Token, therefore a compromised Data Node cannot leak keys owned by other Data Nodes, and attackers cannot access resources beyond the compromised Data Node. Each Container has a unique SEHadoop Child Delegation Token, and every SEHadoop Child Delegation Token can have different access privileges. A compromised Container does not have or has limited impact on data accessed by other Containers. By using unique keys and tokens, SEHadoop improves the isolation level among components of Hadoop.

SEHadoop enforces least access privilege on Hadoop processes when these processes access data in HDFS. SEHadoop

Delegation Token is used to perform authentication and authorization. SplitInfo in SEHadoop Child Delegation Token defines the fine-grained access control information. Each Container has a unique SEHadoop Child Delegation Token to restrict the range of data it can access. Thus a compromised Container can only access the range of data permitted by a SEHadoop Child Delegation Token.

Since several Hadoop components (e.g. Name Node, Resource Manager) manage systemwide sensitive information (e.g. secret keys, user access control policies), compromising of these Hadoop components can cause a systemwide compromise. SEHadoop uses a secure zone to protect these components from various internal cloud attacks. The rest of components of Hadoop still run in a public cloud environment to acquire the benefit of cost saving.

By implementing SEHadoop, compromising small parts of Hadoop only has limited impact on the rest of components in Hadoop, therefore SEHadoop improves compromise resilience of Hadoop in a public cloud. However, if attackers can launch a large scale or systemwide attacks to Hadoop in a public cloud, e.g. compromising cloud management system, SEHadoop only has limited benefits.

IV. SEHadoop Implementation & Experiments

We implemented the SEHadoop model using Hadoop version 2.0.1-alpha. SEHadoop ran on virtual machines (VMs) for the experiments. The guest operating system was Ubuntu 11.04. All VMs ran on VMware ESXi 5.0.0. And VMware ESXi was installed on IBM eServer Blade Center HS21s with two Intel E5450 Xeon CPUs. Each CPU has 4 cores and the clock rate of 3 GHz. The network card was Intel Corporation 82545EM Gigabit Ethernet Controller, and network line-speed was 1 Gbit per second. Both SEHadoop and Hadoop were tested in the same cloud set-up. We used seven VMs for the experiments with each VM having 7 GB available disk space and one CPU core. One VM was the master VM with 6 GB memory. Five VMs were slave VMs with 7 GB memory each. A separated 1 GB memory VM ran Kerberos. The last one VM named Kerberos VM has 1 GB memory. The Name Node, the Resource Manager, and the Job Client ran on the master VM. One Data Node and one Node Manager ran on each slave VM.

We conducted experiments to exhibit how SEHadoop resists to attacks, compare the performances of Hadoop and SEHadoop, and show how to migrate Hadoop jobs to SEHadoop.

We developed two attack scenarios: the HDFS attack, and the YARN attack. Each attack scenario ran on both Hadoop and SEHadoop separately. **The HDFS attack** tried to exploit the overloaded authentication key vulnerability and compromise HDFS. We demonstrated how one compromised Data Node can be used to attack the rest of components of HDFS and how SEHadoop prevents the attack. We used one malicious Data Node to read an HDFS block on another Data Node. The attack code constructed a desired Block, then used the key shared by the Data Node and the Name Node to generate a Block Token. For Hadoop, our experimental results showed that an attacker can successfully fetch the targeted block's content without the Name Node detecting this malicious access. In the SEHadoop, since keys were different on the malicious Data Node and the attacking target Data Node, attack failed. **The YARN attack** aimed to exploit the lack of fine-grained access control vulnerability, and tried to leak data. We exhibited how SEHadoop Delegation Token uses least access privilege principle to restrict the range of data that a compromised Node Manager can access. A malicious Node Manager was used to intercept a Delegation Token, then the Delegation Token was used to access the Delegation Token owner's files. There were two files in HDFS: test1 and test2. Both belonged to the Hadoop test user. The code suppose to read only test1, but the malicious code tried to read the entire content of both test1 and test2 files. In the Hadoop, the malicious code successfully accessed both files. In the SEHadoop, while accessing test2 file, the Name Node rejected the malicious access request and generated the access permission denied exception.

We compared Hadoop's performance with SEHadoop's performance in two cases. In the first case, we examined how much overhead SEHadoop Block Token incurs compared with Hadoop Block Token. In order to clearly compare the performance difference between Hadoop Block Token and SEHadoop Block Token, we minimized operation overhead except for token generation and token verification. We measured the time for a Client to read a small file, 1 byte in size, from both Hadoop HDFS and SEHadoop HDFS. The Client worked with the same Data Node in both Hadoop and SEHadoop scenarios. We used three VMs: Kerberos VM, master VM, and a slave VM. HDFS block replication is one. We repeated to read operation 10,000 times for each system. In the table I, the average times and standard deviations for Hadoop and SEHadoop are showed in seconds. There did not appear to be statistically significant difference between two. The reason was that SEHadoop Block Token used low overhead algorithms, and it generated the same number of tokens as Hadoop did. If the block replication increases, SEHadoop Name Node has to generate one SEHadoop Block Token for each block replication while Hadoop Name Node only needs one Block Token for each block. So we measured the code operation time for SEHadoop Block Token generation and it took 1 to 2 ms in our experiments. SEHadoop Block Token generation time is very small compared with the overall reading time for one byte reading. A large replication of blocks can dramatically reduce the available space of HDFS. HDFS default replication is three. The SEHadoop Block Token generation time is less than 6 ms under the default replication, which is much smaller than the overall reading time. So SEHadoop Block Token performance overhead is still hard to observe in default replication scenario.

We evaluated SEHadoop Delegation Token performance by measuring the time for generating Hadoop Delegation Tokens and SEHadoop Delegation Tokens. We excluded job computing time in the experiment to clearly display performance difference between two Delegation Tokens, due to job computing time varies a lot depending on job code and input data. We measured the time for a Job Client preparing the job and submitting the job to a Resource Manager. For Hadoop, this process includes requesting for a Parent Delegation Token from a Name Node. For SEHadoop, the process includes requesting for a Parent Delegation Token from a Name Node, and generating of Child Delegation Tokens, and etc. The number of map determines how many Child Delegation Tokens need to be generated. We configured the job to generate different number of maps in several cases to show how the number of maps impacts the performance of SEHadoop. The

TABLE I. SEHADOOP BLOCK TOKEN PERFORMANCE

Hadoop	1.806 ± 0.024 (s)
SEHadoop	1.799 ± 0.033 (s)
Overhead Percentage	-0.0039%

TABLE II. SEHADOOP DELEGATION TOKEN PERFORMANCE

No. of Maps	80	640	5120
Hadoop	17.91 ± 0.07 (s)	17.99 ± 0.06 (s)	18.58 ± 0.16 (s)
SEHadoop	17.92 ± 0.09 (s)	18.36 ± 0.10 (s)	21.44 ± 0.28 (s)
Overhead Percentage	0.047%	2.0%	15.4%

randomTextWriter job was used as the performance test job. Table II shows how much time SEHadoop and Hadoop needed in seconds. The number of maps ranged from 80 to 5120. The SEHadoop overhead was comparatively small when there were 80 maps. But the overhead percentage became larger, such as 15.4%, when there were 5120 maps. Hadoop usually uses one map to process one block of data. Default block size is 128 MB. 5120 maps can have more than 655 GB input data. When there were 5120 maps, 2.86 seconds runtime difference between Hadoop Delegation Token and SEHadoop Delegation Token was a comparative small comparing to the overall job executing time to process 655 GB data which usually needs more than a half hour. The measured operations were running on the Job Client which is outside of SEHadoop computing cluster. So even for a Hadoop job with a large number of maps, SEHadoop Delegation Token has very limited performance impact on overall Hadoop job executing progress.

Our experiments exhibit that all Hadoop jobs can run on SEHadoop. Migrating Hadoop jobs to SEHadoop is straightforward. We customized the splitter to set a proper range for each FileSplit.

V. RELATED WORK

We believe that we are the first to offer a low-overhead compromise resilience solution to Hadoop in a public cloud. We are not aware of any published discussion of the impact of low isolation level and the lack of fine-grained access control in Hadoop. The overloaded authentication keys vulnerability was discussed in the Hadoop security design report [1], but the authors of that report were relatively dismissive of its impact. In general, a number of of researchers investigated at improving security of Hadoop running in a cloud. Myers and Abdelnur implemented encrypted channel for Hadoop various communications [16], [17]. Tahoe [18] encrypts data in a distributed file system and performs integrity-check on its gateway. HIMA [14] is hypervisor-based agent to measure integrity of user programs in VM which can improve overall security of a cloud environment.

VI. CONCLUSION

We studied the potential internal cloud attacks to Hadoop in a public cloud. We designed and implemented SEHadoop model that consists of SEHadoop runtime model, SEHadoop Block Token and SEHadoop Delegation Token to improve compromise resilience of Hadoop in a public cloud. SEHadoop model enhances isolation level among Hadoop components and enforces least access privilege on Hadoop processes. Our experimental results exhibit how enhanced isolation and least access privilege of SEHadoop prevent attackers from using compromised Hadoop processes to compromise the rest of components of Hadoop. SEHadoop Block Token does not appear to inflict overhead, and SEHadoop Delegation Token has very limited performance impact. The experiment result also showed migrating Hadoop jobs to SEHadoop is straightforward.

ACKNOWLEDGEMENT

This work is supported by National Science Foundation grant 0910767 and 1330533, U.S.A.

REFERENCES

[1] O. O'Malley, K. Zhang, S. Radia, R. Marti, and C. Harrell, "Hadoop security design," *Yahoo, Inc., Tech. Rep*, 2009.

[2] Joyent, "Joyent solution for hadoop," http://www.joyent.com/products/computeservice/ features/hadoop/. Accessed in November 2014.

[3] Inforchimps, "Cloud::hadoop," http://www.infochimps.com/infochimps-cloud/cloud- services/cloud-hadoop/ Accessed in Nov. 2014.

[4] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 583 – 592, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S01677-39X10002554

[5] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 1 – 11, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S10848-04510001281

[6] "Top threats to cloud computing," https://cloudsecurityalliance.org/topthreats/ csathreats.v1.0.pdf Accessed in April 2014.

[7] K. Kortchinsky, "Cloudburst: A VMware guest to host escape story," http://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloud burst-PAPER.pdf. Accessed in Nov. 2014.

[8] J. R. R. Wojtczuk, "Xen 0wning trilogy," in Black Hat Conf., 2008.

[9] Secunia, "Vulnerability report: VMware esx server 3.x," http://secunia.com/advisories/product/10757/. Accessed in November 2014.

[10] S. Advisory, "Xen pv kernel decompression multiple vulnerabilities," http://secunia.com/advisories/44502/. Accessed in November 2014.

[11] F. Rocha and M. Correia, "Lucy in the sky without diamonds: Stealing confidential data in the cloud," in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*. IEEE, 2011, pp. 129–134.

[12] S. Bugiel, S. Nürnberger, T. Pöppelmann, A.-R. Sadeghi, and T. Schneider, "Amazonia: when elasticity snaps back," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 389–400.

[13] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 199–212.

[14] A. Azab, P. Ning, E. Sezer, and X. Zhang, "Hima: A hypervisor-based integrity measurement agent," in *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, 2009, pp. 461–470.

[15] Y. Zhang and M. K. Reiter, "Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 827–838.

[16] A. T. Myers, "Add support for encrypting the datatransferprotocol," https://issues.apache.org/jira/browse/HDFS-3637. Accessed in November 2014.

[17] A. Abdelnur, "Add support for encrypted shuffle," https://issues.apache.org/jira/browse/ MAPREDUCE-4417. Accessed in November 2014.

[18] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*, ser. StorageSS '08. New York, NY, USA: ACM, 2008, pp. 21–26. [Online]. Available: http://doi.acm.org/10.1145/1456469.1456474