International Journal For Technological Research In Engineering
Volume 2, Issue 9, May-2015 *ISSN (Online): 2347 - 4718*

A REVIEW ON HADOOP SECURITY AND RECOMMENDATIONS

Dr. S. R. Gupta[1], Apeksha G. Dengre[2]
[1]Assistant Professor, Computer Science & Engineering, PRMIT, Maharashtra, India
[2](M.E Scholar), Computer Science & Engineering, PRMIT, Maharashtra, India

Abstract: Hadoop is a distributed system that provides a distributed file system and MapReduce batch job processing on large clusters using commodity servers. Although Hadoop is used on private clusters behind an organization's firewalls, Hadoop is often provided as a shared multi-tenant service and is used to store sensitive data; as a result, strong authentication and authorization is necessary to protect private data. Adding security to Hadoop is challenging because all the interactions do not follow the classic client server pattern. To address these challenges, the Hadoop community decided to focus on authentication, and chose the base Kerberos authentication mechanism which is supplemented by delegation and capability-like access tokens and the notion of trust for secondary services.
Keywords: Hadoop, HDFS, MapReduce, Kerberos

I. INTRODUCTION

One of the biggest concerns in our present age revolves around the security and protection of sensitive information. In our current era of Big Data, our organizations are collecting, analyzing, and making decisions based on analysis of massive amounts of data sets from various sources, and security in this process is becoming increasingly more important. At the same time, more and more organizations are being required to enforce access control and privacy restrictions on these data sets to meet regulatory requirements such as HIPAA and other privacy protection laws. Network security breaches from internal and external attackers are on the rise, often taking months to be detected, and those affected are paying the price. Organizations that have not properly controlled access to The more data you have, the more important it is that you protect it. It means that not only must we provide effective security controls on data leaving our networks, but we also must control access to data within our networks. Apache Hadoop is a distributed system for storing large amounts of data and processing the data in parallel. Hadoop is used as a multi-tenant service internally at Yahoo! and stores sensitive data such as personally identifiable information or financial data. Other organizations, including financial organizations, using Hadoop are beginning to store sensitive data on Hadoop clusters. As a result, strong authentication and authorization is necessary. This paper describes the security issues that arise in Hadoop and how we address them. Hadoop is a complex distributed system that poses a unique set of challenges for adding security. While we rely primarily on Kerberos as our authentication mechanism, we supplement it with delegation tokens, capability-like access tokens and the notion of trust for auxiliary services.

A. Hadoop Background

Hadoop has been under development at Yahoo! And a few other organization as an Apache open source project over the last 5 years. It is gaining wide use in the industry. Yahoo!, for example, has deployed tens of Hadoop clusters, each typically with 4,000 nodes and 15 petabytes. Hadoop contains two main components. The first component, HDFS [8], is a distributed file system similar to GFS [4]. HDFS contains a metadata server called the NameNode that stores the hierarchical file and directory name space and the corresponding metadata, and a set of DataNodes that stores the individual blocks of each files. Each block, identified by a block id, is replicated at multiple DataNodes. Client perform file metadata operations such as create file and open file, at the NameNode over an RPC protocol and read/write the data of a file directly to DataNodes using a streaming socket protocol called the data-transfer protocol.

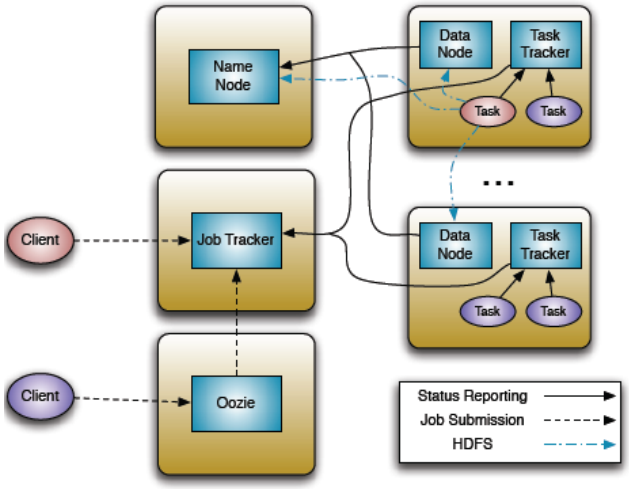

Figure 1: Hadoop High-level Architecture

The second component is a framework for processing large amounts of data in parallel using the MapReduce paradigm [2] . HDFS DataNodes also serve as compute nodes for MapReduce to allow computation to be performed close to the data being processed. A user submits a MapReduce job to the JobTracker which schedules the job to be executed on the compute nodes. Each compute node has a small daemon called the TaskTracker that launches map and reduce tasks of a job; the task tracker also serves intermediate data created by map tasks to reduce tasks. There are additional services deployed with Hadoop, one of which are relevant to the security concerns discussed in this paper. Ooozie is a workflow system that provides a way to schedule and submit a DAG of MapReduce jobs that are triggered for execution by data availability or time.

II. CHALLENGES IN ADDING SECURITY TO HADOOP

Adding security to Hadoop is challenging in that not all interactions follow the usual client-server pattern where the server authenticates the client and authorizes each operation by checking an ACL:

1. The scale of the system offers its own unique challenges. A 4000 node typical cluster is expected to serve over 100,000 concurrent tasks; we expect the cluster size and the number of tasks to increase over time. Commonly available Kerberos [9] servers will not be able to handle the scale of so many tasks authenticating directly.
2. The file system is partitioned and distributed: the NameNode processes the initial open/create file operations and authorizes by checking file ACLs. The client accesses the data directly from the DataNodes which do not have any ACLs and hence have no way of authorizing accesses.
3. A submitted batch job is executed at a later time on nodes different from the node on which the user authenticated and submitted the job. Hence the users authentication at the job submission computer needs to be propagated for later use when the job is actually executed (note the user has usually disconnected from the system when the job actually gets executed.) This propagated credentials raises issues of trust and offers opportunities for security violation.
4. The tasks of a job need to securely obtain information such as task parameters, intermediate output of tasks, task status, etc. Intermediate Map output is not stored in HDFS; it is stored on the local disk of each compute node and is accessed via an HTTP-based protocol served by the Task-trackers.
5. Tasks from different tenants can be executed on the same machine. This shared environment offers opportunities for security violations via the APIs of the local operating system of the compute node: access to the intermediate output of other tenants, access to concurrent tasks of other jobs and access to the HDFSs local block storage on the node.
6. Users can access the system through auxiliary service such as Oozie, our work flow system. This raises new issues. For example, should the users credentials be passed through these secondary services or should these services be trusted by the rest of the Hadoop system? Some of these problems are unique to systems like Hadoop. Others, as individual problems, occur in other systems. The overall set of solutions and how they interact, to our knowledge are fairly unique and instructive to a designer of complex distributed systems.

Many of these can currently be answered by Hadoop's current capabilities, but many of them cannot, leading to the proliferation of Hadoop security-complementing tools that we see in the industry. Just a few reasons that vendors are releasing security products that complement Hadoop are:

1. No "Data at Rest" Encryption. Currently, data is not encrypted at rest on HDFS. For organizations with strict security requirements related to the encryption of their data in Hadoop clusters, they are forced to use third-party tools for implementing HDFS disk-level encryption, or security-enhanced Hadoop distributions (like Intel's distribution from earlier this year).
2. A Kerberos-Centric Approach – Hadoop security relies on Kerberos for authentication. For organizations utilizing other approaches not involving Kerberos, this means setting up a separate authentication system in the enterprise.
3. Limited Authorization Capabilities – Although Hadoop can be configured to perform authorization based on user and group permissions and Access Control Lists (ACLs), this may not be enough for every organization. Many organizations use flexible and dynamic access control policies based on XACML and Attribute-Based Access Control. Although it is certainly possible to perform these level of authorization filters using Accumulo, Hadoop's authorization credentials are limited
4. Complexity of the Security Model and Configuration. There are a number of data flows involved in Hadoop authentication – Kerberos RPC authentication for applications and Hadoop Services, HTTP SPNEGO authentication for web consoles, and the use of delegation tokens, block tokens, and job tokens. For network encryption, there are also three encryption mechanisms that must be configured – Quality of Protection for SASL mechanisms, and SSL for web consoles, HDFS Data Transfer Encryption. All of these settings need to be separately configured – and it is easy to make mistakes.

A. Architectural Issues

• Distributed nodes: "Moving computation is cheaper than moving data" is the key to big data. Data is processed anywhere resources are available, enabling massively parallel computation. This creates complicated environments with plenty of attack surface, and it is difficult to verify security consistency across a highly distributed cluster of possibly heterogeneous platforms.

• 'Sharded' data: Data within big data clusters is fluid, with multiple copies moving to and from different nodes to ensure redundancy and resiliency. A 'shard' is a slice of data — horizontally segmented — shared across multiple servers. This automated movement to multiple locations makes it very difficult to know precisely where data is located at any moment in time, or how many copies are available. This runs counter to the traditional centralized data security model, where a single copy of data is wrapped in various protections until it is used for processing. Big data is replicated in many places and moves as needed. The 'containerized' data security model is missing — as are many other relational database facilities.

• Data access/ownership: Role-based access is central to most database security schemes. Relational and quasirelational platforms include roles, groups, schemas, label security, and various other facilities for limiting user access to authorized subsets of the available data. Most big data environments offer access limitations at the schema level, but no finer granularity. It is possible to logically mimic label security and other advanced capabilities in big data environments, but that requires the application designer to build these functions into applications and data storage.

• Inter-node communication: Hadoop and the vast majority of distributions don't communicate securely — they use RPC over TCP/IP. TLS and SSL are rarely bundled in the big data distribution. When they are — as with HDFS proxies — they

only cover client-to-proxy communication, not proxy-to-node communication.

• Client interaction: Clients interact with resource managers and nodes. While gateway services can be created for loading data, clients communicate directly with both resource managers and individual data nodes. Compromised clients can send malicious data or links to either service. This model facilitates efficient communication but makes it difficult to protect nodes from clients, clients from nodes, or even name servers from nodes. Worse, the distribution of self-organizing nodes is a poor fit for security tools such as gateways/firewalls/monitoring, which require a 'choke-point' not available in a peer-to-peer 'mesh' cluster.

• NoSecurity: Finally, and perhaps most importantly, big data stacks build in almost no security. As of this writing —aside from service-level authorization and web proxy capabilities from YARN — no facilities are available to protect data stores, applications, or core Hadoop features. All big data installations are built on the web services model, with few or no facilities for countering common web threats, (i.e. everything on the OWASP Top Ten list) so most big data APIs are vulnerable to well known attacks.

Attacks on corporate IT systems and data theft are prevalent, and large distributed data management systems provide a tempting target. Big data offers all the same weak points we know from traditional IT systems, but we have technologies that address common threats so there is no need to reinvent the wheel. The trick is selecting options that work with big data. Cluster administrators should consider security controls for each of the following areas when setting up or managing a big data cluster:

• Data at rest protection: The standard for protecting data at rest is encryption, which guards against attempts to access data outside established application interfaces. With traditional data management systems we worry about people stealing archives or directly reading files from disk. Encrypted files are protected against access from users without encryption keys. Replication effectively replaces backups for big data, but that does not mean a rogue administrator or cloud service manager won't create their own. Encryption protects data copied from the cluster. One or two obscure NoSQL variants provides encryption for data at rest but most do not. Worse, most available encryption products lack sufficient horizontal scalability and transparency to work with big data. This is a critical issue.

• Administrative data access: Each node has at least one administrator with full access to its data. As with encryption we need a boundary or facility to provide separation of duties between different administrators. The requirement is the same as on relational platforms — but big data platforms lack their array of built-in facilities, documentation, and thirdparty tools to address this requirement. Unwanted direct access to data files or data node processes can be addressed through a combination of access controls, separation of roles, and encryption technologies — but out-ofthe- box, data is only as secure as your least trustworthy administrator. It's up to the system designer to select controls to close this gap.

• Configuration and patch management: With clusters of servers it is common to have nodes running different configurations and patch levels as new nodes are added over time. Or if you use dissimilar OS platforms in the cluster determining what constitutes equivalent patch revision levels can be difficult. Existing configuration management tools work for underlying platforms, and HDFS Federation will help with cluster management, but careful planning is still necessary. We will go into detail about how to accomplish this later, under Recommendations. The cluster may tolerate nodes cycling without loss of data or service interruption but reboots can still cause serious performance issues, depending on which nodes are affected and how the cluster is configured. The upshot is that people worry about user complaints and don't patch. Perhaps you have heard that one before.

• Authentication of applications and nodes: Hadoop can use Kerberos to authenticate users and add-on services to the Hadoop cluster. But a rogue client can be inserted onto the network if a Kerberos ticket is stolen or duplicated — perhaps using credentials extracted from virtual image files or snapshots. This is more of a concern when embedding credentials in virtual and cloud environments, where it is relatively easy to introduce an exact replica of a client app or service. A clone of a node is often all that's needed to introduce a corrupted node or service into a cluster. It is easy to impersonate a clust er node or service, but that requires an attacker to compromise your management plane or obtain a client backup. Kerberos improves security significantly but you still need to be careful. We know it is a pain to set up, but strong authentication of nodes is a principal security tool for keeping rogue servers and requests out of your cluster.

• Audit and logging: If you suspect someone has breached your cluster, can you detect it? How could you do this? You need a record of activity. One area which offers a variety of add-on capabilities is logging. Scribe and LogStash are open source tools that integrate into most big data environments, as do a number of commercial products. So you just need to find a compatible tool, install it, integrate it with other systems such as SIEM or log management, and then actually review the results. Without actually looking at the data and developing policies to detect fraud, logging is not useful.

• Monitoring, filtering, and blocking: There are no built-in monitoring tools to look for misuse or block malicious queries. In fact there is no consensus on what a malicious big data query looks like — aside from crappy MapReduce scripts written by bad programmers. It is assumed that you will authenticate clients through Kerberos if you care about security, and MapReduce access is gated by digest authentication. Several monitoring tools are available for big data environments, but most review data and user requests at the API layer. The problem is that these solutions require a 'choke-point' — a path through which all client connections must pass. Our own David Mortman called these "aftermarket speed regulators" because the security bottleneck inherently limits performance. Most deliver the basic security value they claim, but require you to alter your deployment model or to forgo scaling — or both.

• API security: The APIs for big data clusters need to be

protected from code and command injection, buffer overflow attacks, and every other web services attack. Most of the time this responsibility falls upon the application(s) that use the cluster, but not always. Common security controls include integration with directory services, mapping OAuth tokens to API services, filtering requests, input validation, managing policies across nodes, and so on. Some of the APIs even work without authentication, so people still haven't yet acknowledged the problem. Again, there are a handful of off-the-shelf solutions to help address API security issues, but most are based on a gateway that funnels users and all requests through a single interface. There are many important API issues, but they are beyond the scope of this paper. In summary, there are many ways to build out a big data cluster, but few relevant and usable security tools. You can address the most glaring security weaknesses with a combination of built-in authentication services and add-on security products, without wrecking performance or scalability.

III. SECURITY REQUIREMENTS

Any security control used for big data must meet the following requirements:

1. It must not compromise the basic functionality of the cluster.
2. It should scale in the same manner as the cluster.
3. It should not compromise essential big data characteristics.
4. It should address a security threat to big data environments or data stored within the cluster.

IV. SECURITY RECOMMENDATIONS

In the end, our big data security recommendations boil down to a handful of standard tools which can be effective in setting a secure baseline for big data environments:

1. Use Kerberos for node authentication: Kerberos is effective for validating inter-service communicate and help keep rogue nodes and applications out of the cluster. And it can help protect web console access, making administrative functions harder to compromise. We know Kerberos is a pain to set up, and (re-)validation of new nodes and applications requires additional overhead. But without bi-directional trust establishment, it is too easy to fool Hadoop into letting malicious applications into the cluster or accepting malicious nodes — which can then add, alter, and extract data. Kerberos is one of the most effective security controls at our disposal, and it's built into the Hadoop infrastructure, so use it.
2. Use file layer encryption: File encryption protects against two attacker techniques for circumventing application security controls. Encryption protects data if malicious users or administrators gain access to data nodes and directly inspect files, and renders stolen files or copied disk images unreadable. While it may be tempting to rely upon encrypted SAN/NAS storage devices, they don't provide protection from credentialed user access, granular protection of files or multi-key support. And file layer encryption provides consistent protection across different platforms regardless of OS/platform/storage type, with some products even protecting encryption operations in memory. Just as important, encryption meets our requirements for big data security — it is transparent to both Hadoop and calling applications, and scales out as the cluster grows. Open source products are available for most Linux systems; commercial products additionally offer external key management, trusted binaries, and full support. This is a cost-effective way to address several data security threats.
3. Use key management: File layer encryption is not effective if an attacker can access encryption keys. Many big data cluster administrators store keys on local disk drives because it's quick and easy, but it's also insecure as keys can be collected by the platform administrator or an attacker. Use key management service to distribute keys and certificates; and manage different keys for each group, application, and user. This requires additional setup and possibly commercial key management products to scale with your big data environment, but it's critical. Most of the encryption controls we recommend depend on key/certificate security.
4. Deployment validation: Deployment consistency is difficult to ensure in a multi-node environment. Patching, application configuration, updating the Hadoop stack, collecting trusted machine images, certificates, and platform discrepancies all contribute to what can easily become a management nightmare. The good news is that most big data clusters are deployed in cloud and virtual environments. You can leverage tools from your cloud provider, your hypervisor vendor, and third parties (such as Chef and Puppet) to automate pre-deployment tasks. Machine images,
patches, and configurations should be fully updated and validated prior to deployment. You can even run validation tests, collect encryption keys, and request access tokens before nodes are accessible to the cluster. We also recommend use of the service-level authentication built into Hadoop to help segregate administrative responsibilities. Building the scripts and setting up these services takes time up front, but pays for itself in reduced management time and effort later, and ensures that each node comes online with baseline security in place.
5. Log it! : To detect attacks, diagnose failures, or investigate unusual behavior, you need a record of activity. Unlike less scalable data management platforms, big data is a natural fit for collecting and managing event data. Many web companies start with big data specifically to manage log files, and most SIEM and log management products embed big data capabilities for log management. There is no reason not to add logging onto your existing cluster. It gives you a place to look when something fails, or if someone thinks you might have been hacked. Without an event trace you are blind. Logging MapReduce requests and other cluster activity is easy, and the increase in storage and processing demands is small, but the data is indispensable when you need it.
6. Use secure communication: Implement secure communication between nodes, and between nodes and applications. This requires an SSL/TLS implementation that actually protects all network communications rather than just a subset. This imposes a small performance penalty on transfer of large data sets around the cluster, but the burden

is shared across all nodes. Cloudera offers TLS, and some cloud providers offer secure communication options as well; otherwise you will likely need to integrate these services into your application stack. These are easy to recommend — they are simple, cost-effective, and scalable, and they addresses real security deficiencies with big data clusters. Nothing suggested here harms performance, scalability, or functionality.

V. CONCLUSION

Hadoop, the system and its usage grew over the last 5 years. The early experimental use did not require security and there weren't sufficient resources to design and implement it. However as Hadoop's use grew massively, security became critical. Hadoop's elastic allocation of resources and the scale at which it is is typically deployed lends Hadoop to be shared across tenants where security is critical if any stored data is sensitive. Segregating sensitive data and customers into private clusters was not practical or cost effective. As a result, security was recently added to Hadoop in spite of the axiom that states it is best to design security in from the beginning. The adoption of Hadoop in commercial organizations is beginning and security is a critical requirement. In our fast-paced and connected world where Big Data is king, it is critical to understand the importance of security as we process and analyze massive amounts of data. This starts with understanding our data and associated security policies, and it also revolves around understanding the security policies in our organizations and how they need to be enforced. This paper provided a brief history of Hadoop Security, focused on common security concerns, and ways to address them.

REFERENCES

[1] Andrew D. Birrell. Secure communication using remote procedure calls. ACM Trans. Comput. Syst., 3:1–14, February 1985.
[2] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI, pages 137– 150, 2004.
[3] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In Proceedings of the 5th ACM Conference 10 on Computer and Communications Security Conference, pages 83–92, 1998.
[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.
[5] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. ACM Trans. Comput. Syst., 6:51– 81, February 1988.
[6] Zach Miller, Dan Bradley, Todd Tannenbaum, and Igor Sfiligoi. Flexible session management in a distributed environment. Journal of Physics: Conference Series, 219(4):042017, 2010.
[7] Mahadev Satyanarayanan. Integrating security in a large distributed system. ACM Trans. omput. Syst., 7(3):247– 280, 1989.
[8] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on, 0:1–10, 2010.
[9] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In in Usenix Conference Proceedings, pages 191–202, 1988.
[10] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience, 17(2- 4):323–356, 2005.
[11] Victor L. Voydock and Stephen T. Kent. Security mechanisms in high-level network protocols. ACM Comput. Surv., 15(2):135–171, 1983
[12] Lee Hyeokju, Kim Myoungjin, Joon Her, and Hanku Lee,Division of Internet & Multimedia Engineering, Konkuk University Seoul Korea in 2011 Ninth IEEE International Conference on "Dependable, Autonomic and Secure Computing".
[13] Sun-Moo Kang, Bu-Ihl Kim, Hyun-Sok Lee, Young-so Cho, Jae-Sup Lee, Byeong-Nam Yoon, "A study on a public multimedia sevice provisioning architecture for enterprise networks", Network Operations and Management Symposium, 1998, NOMS 98., IEEE, 15-20 Feb 1998, 44-48 vol.1, ISBN : 0 -7803-4351-4

