International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V72I9P103

Volume 72 Issue 9, 33-48, September 2024
© 2024 Seventh Sense Research Group®

Original Article

Preliminary Analysis of HDFS Read Operation, Threats Impacts and Mitigation

Imane LEBDAOUI[1], Ghizlane ORHANOU[2]

[1,2]*Laboratory of Mathematics, Computing and Applications Information Security (LabMiA-SI), Faculty of Sciences, Mohammed V University in Rabat, Rabat, Morocco.*

[1]*Corresponding Author : imane.lebdaoui@gmail.com*

Received: 25 April 2024 Revised: 11 July 2024 Accepted: 12 August 2024 Published: 28 September 2024

Abstract - *The Hadoop Distributed File System (HDFS) is widely used to store and enable access to, read, and write large volumes of data and files. However, HDFS, like Hadoop, remains vulnerable to numerous security threats that make it a target for malicious activities, leading to the loss and manipulation of data and files in an illegal manner. To this end, this research aims to reconsider the preservation of security throughout the operation of reading files from HDFS, focusing on preserving the security of the assets and flows involved. Using a systematic security analysis of the elements mentioned above, this article addresses the existing security issues, discusses the security requirements to be met to enable better file reading from HDFS, and analyzes the impact of various threats on the associated assets. Through the analysis of three main attack use cases, the threats are identified and classified into six threat families. For each family of threats, mitigation measures are proposed to reduce the impact of the threats and enable better reading operations from HDFS.*

Keywords - *HDFS Read operation, Hadoop security, Threat analysis, Mitigation, DataNodes.*

1. Introduction

HDFS serves as the storage layer for Hadoop and plays a crucial role in the Hadoop ecosystem, providing foundational support for other Hadoop applications and external systems or users. It is designed to store large files that are frequently accessed by users or systems. In HDFS, reading operations are more common than writing, as files are typically written once but read many times by multiple systems or users. Consequently, this paper focuses on the process of reading files and data from HDFS. The primary goal of a big data storage system like HDFS is to return accurate data in a timely manner while ensuring its credibility, quality, and timeliness. In HDFS, large files and data are divided into multiple blocks, which are replicated across various nodes [6]. This fragmentation ensures data availability even if one or more servers fail. However, ensuring other aspects of security is essential to maintain the quality and credibility of the files and data. The Hadoop framework and HDFS lack sufficient security measures to fully protect stored files against various threats [1, 2, 5, 7]. These security gaps can affect the ability to read and access files stored in HDFS effectively.

Therefore, a thorough security analysis is essential for identifying and addressing vulnerabilities and weaknesses associated with HDFS read operations. This analysis provides a comprehensive foundation for minimizing and mitigating potential security risks.

1.1. Motivations

This paper specifically analyzes the read operation in HDFS, including its ecosystem and involved components. By examining the interactions between these components during the read operation, the paper identifies the primary threats and discusses the key security use cases. Following a logical sequence, the analysis begins by defining the assets involved and presenting the associated Data Flow Diagram (DFD). It then provides an in-depth examination of the HDFS read operation. The analysis continues with a discussion of existing security issues, highlighting their implications for the assets involved in the read operation. The ultimate goal is to propose effective mitigation actions.

1.2. Contributions

Our contributions are as follows:

Detailed Analysis of HDFS Read Operations: We provide a comprehensive description of the crucial operation of reading files from HDFS, which stores large volumes of data that are accessed continuously and intensively. Given the limited coverage of this topic in existing literature, we address this gap by detailing the processes and actors involved, identifying various entry points, and describing the data exchange flows and communication channels. Security Requirements and Issues: We examine the security requirements for all components and assets involved in HDFS read operations. This includes addressing security concerns related to data and data storage, network exchanges, and system processes. The paper also analyzes the threats and security issues that could influence the operation of reading files from HDFS and discusses relevant security use cases.

1.3. Outline

The paper is structured as follows:

Section 2: Provides technical background on HDFS components, including their roles, structures, and interactions.

Section 3: Offers an overview of HDFS read operation mechanisms, detailing the steps involved, data flows, and communications between components.

Section 4: Presents an architectural design of the HDFS read operation through a Data Flow Diagram (DFD), tracing data, requests, and exchanges. It also defines four asset families, maps security requirements for each, and reviews related works on HDFS security.

Section 5: Analyzes three critical use cases selected based on the previous sections. Each use case is detailed through attack scenarios and affected assets, followed by proposed measures to mitigate the associated threats.

Section 6: Concludes with a summary of results and future perspectives.

2. Technical Background

2.1. HDFS Structure and Operation

HDFS (Hadoop Distributed File System) is a distributed file system for storing very large files, operating on clusters of standard hardware [5]. It is designed for low-cost, scalable hardware, extremely simple to expand, and highly tolerant of defects.

HDFS follows Write Once Read Many modes, where the data is written on the server once and read and reused numerous times after that [8]. Read operation corresponds to downloading files from the distributed file system [6]. HDFS operates in data replication mode in which multiple blocks of data are replicated and distributed across nodes in a cluster. This replication ensures the high availability of data in case of a node failure. Several features make HDFS particularly useful and robust, including:

2.1.1. Data Replication

In HDFS, DataBlocks are typically replicated across many nodes. The number of replicas, known as the replication factor, is reconfigurable. Replication ensures that data is always available and prevents data loss.

Thus, the data is always accessible even in case of a DataNode crash or a data block damage. Replication implies fault tolerance and reliability of the data in HFDS and ensures their high availability.

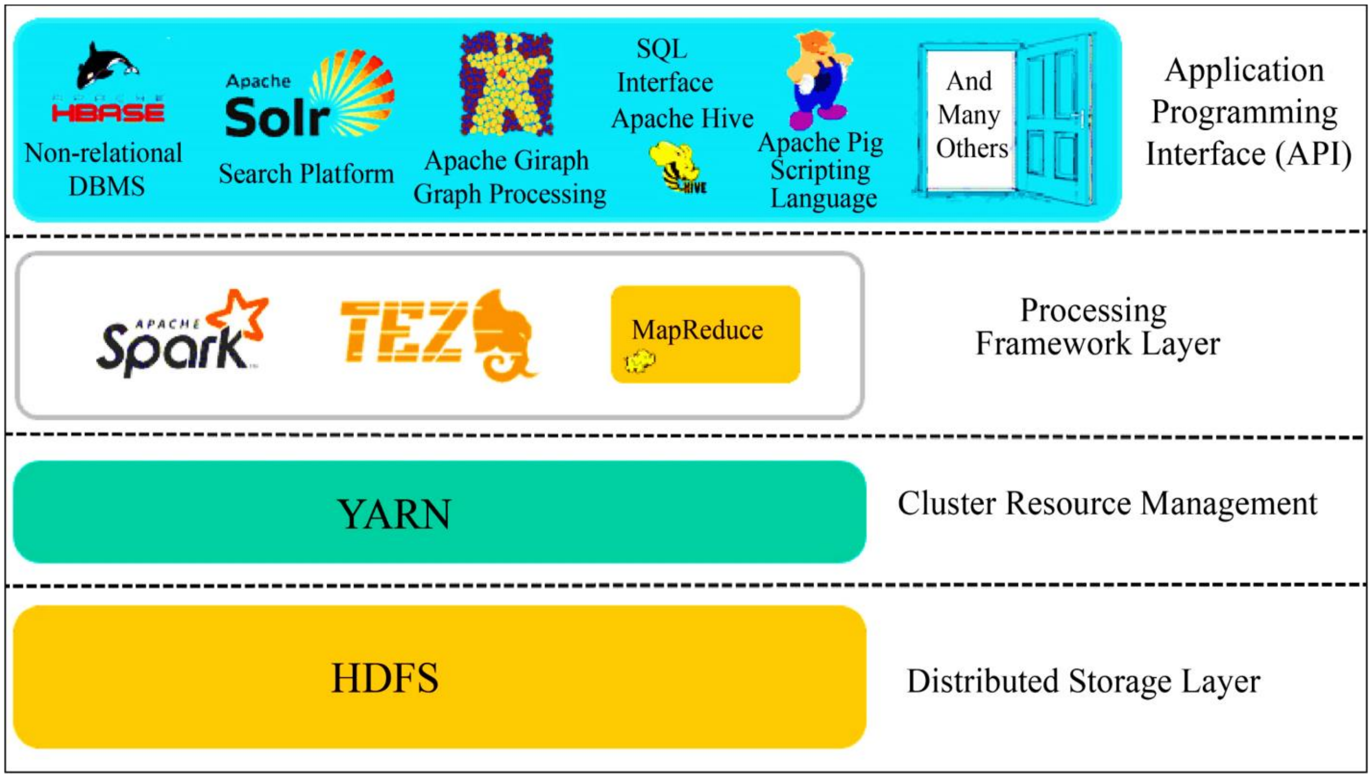
HBASE
Non-relational DBMS
Apache Solr
Search Platform
Apache Giraph
Graph Processing
SQL Interface Apache Hive
HIVE
Apache Pig Scripting Language
And Many Others
Application Programming Interface (API)
APACHE Spark
TEZ
MapReduce
Processing Framework Layer
YARN
Cluster Resource Management
HDFS
Distributed Storage Layer

Fig. 1 Hadoop ecosystem layers [4]

2.1.2. Scalability

Since HDFS stores data on various nodes in the cluster, “A cluster can expand to hundreds of nodes as demand grows.

2.1.3. High Throughput

The storage of the data in a distributed manner enables their processing in parallel on a cluster of nodes. Throughput is consequently improved.

2.1.4. Data Locality

With HDFS, computation takes place directly on the DataNodes where the data is stored, eliminating the need to transfer data to the computational units. In fact, HDFS offers interfaces that allow applications to position themselves closer to the data**.** [9]. This approach decreases network congestion and boosts a system’s throughput.

2.1.5. Portability

HDFS is designed for easy portability across different platforms [9], facilitating its adoption as the preferred choice for a wide range of applications.

2.1.6. Cluster Rebalancing

The HDFS architecture supports data rebalancing schemes that automatically move data between DataNodes when the free space on a DataNode falls below a certain threshold [9].

2.2. HDFS Components Role

HDFS operates using master-slave or master-worker topology [10], where the master machine runs a daemon called NameNode (NN), and the enslaved people or workers’ machines run DataNodes (DN). Three components constitute HDFS: HDFS Client, NameNode and DataNodes.

2.2.1. HDFS Client

In Hadoop, Client refers to the interface used to communicate with the Hadoop Filesystem. Many types of Clients are available in Hadoop to perform different tasks. These Clients can be invoked using their respective CLI (Command Line Interface) commands from a node where Hadoop is installed and has the necessary configurations and libraries required to connect to a Hadoop Filesystem. On the other hand, the HDFS Client (or Hadoop Client) is an intermediate component between the HDFS and the user, and it interacts, on the user’s behalf, with NameNode and DataNodes to fulfill user requests. It is responsible for chunking a file in blocks of 128 MB (in Hadoop 2) that it saves in HDFS. In addition, It supports the processes of reading, writing and deleting files [11].

2.2.2. NameNode

The NameNode (NN) is the core of HDFS and is considered the HDFS central controller [11]. It tracks where, across the cluster, the file data resides, and it maintains the hierarchical file tree structure and the metadata for all files and directories in the system [5]. It also stores “File to Block(s) mapping” MetaData. However, it does not store the data contained in these files locally [11, 12] but stores them in the Hadoop FileSystem. It does not store either block location (“Block to Datanode(s) mapping” MetaData) persistently, and it is possible to reconstruct this information from the DataNodes once the system starts. This type of MetaData is then stored in memory for the read and write operations. Two types of files, ‘FileSystem Image’ and the ‘edit logs’, are used to store MetaData information. More details about these two types of files, their content and their use will be given in subsection 3.1 [13]. The NameNode constitutes the arbitrator and repository for all HDFS MetaData (stored in disk or memory), which is used for file read and write operations. It never calls DataNodes directly, but instead, it delivers commands in response to DataNodes heartbeats.
These commands include [5]:

- Distribute blocks to other nodes
- Remove local block replicas
- Re-register the node or shut it down
- Immediately submit a block report.

The NameNode can handle thousands of heartbeats per second without disrupting its other operations.

2.2.3. DataNode

DataNodes are the worker nodes in HDFS, responsible for storing and retrieving blocks as directed by Clients or the NameNode. They periodically report to the NameNode with lists of the blocks they are storing [5, 14]. Each data block in DataNode has two parts stored in separate files: data itself and its MetaData (checksums for the data block and its generation stamp). Each DataNode submits a block report to the NameNode that includes the block ID, generation stamp, and length for each block replica it hosts. The initial report is sent immediately upon DataNode registration, and subsequent reports are sent every hour to provide the NameNode with an updated view of block replica locations within the cluster. DataNodes also handle block creation, deletion, and replication based on instructions from the NameNode.

2.2.4. HDFS Access and Components Interaction

There are generally three ways to access the HDFS filesystem [6, 9]:

- FileSystem Shell (FSS) is used through the CLI of the “built-in” client, which is provided together with the Hadoop distribution. FSS is implemented in the HDFS program, situated in the Hadoop/bin directory. To read a file filexx.txt, a user types bin/Hadoop dfs-cat /mydir/filexx.txt.

- The HTTP REST API interface provided by Web-HDFS will be easier than the native interface. The web API allows external applications to interact with the HDFS and read/write files. However, the HTTP interface is slower than the native Java client, especially for large data transmission.
- By DFS Admin, which is used to perform administration commands.

Furthermore, The NameNode and DataNode are software components designed to operate on commodity hardware [5, 9, 15]. The DataNode talks to the NameNode using the DataNode Protocol, as illustrated in Figure 2. It periodically sends a Heartbeat message to the NameNode (the default heartbeat interval is three (3) seconds). It marks DataNodes that have not sent a heartbeat within 10 minutes as dead and ceases to forward new I/O requests to them [9].

HDFS communication protocols are layered on top of the TCP/IP protocol stack [9]:

A client connects to a configurable TCP port on the NameNode and uses the Client Protocol, as defined in ClientProtocol.java [15]. The DataNode communicates with the NameNode using the DataNode Protocol. A Remote Procedure Call (RPC) abstraction wraps both the Client Protocol and the DataNode Protocol. A Client communicates with a DataNode directly to transfer (send/receive) data using the DataTransferProtocol, defined in DataTransferProtocol.java.

For performance purposes, this protocol is not RPC; it is based on a streaming protocol. HDFS forms the storage layer and basis of Hadoop. This article also zoomed in on the components of HFDS and showed how they interact with each other. These interactions enable data access, data reading, data writing, data replication, and data storage.

In the following sections, this article focuses on the read operations that continuously occur in HDFS and examines the various information flows associated with these operations. The objective is to identify threats that may impact the smooth functioning of data reading from HDFS.

3. Read Mechanism in HDFS

Each operation on a file or in a directory passes the full path name to the NameNode, and the permissions checks are applied along the path for each operation [9]. In HDFS, files are divided into blocks of configurable size, with each block stored as an independent unit.

The default block size in HDFS 2 is 128 MB, This size can also be reconfigured to become higher or lower according to requirements. The NameNode knows which DataNode contains which blocks thanks to “Block to DataNode(s) mapping” MetaData stored in memory and also where the DataNodes reside within the machine cluster [8, 13].The following section is focused on the HDFS read operation, the data and data stores used, and the interactions between different components involved in the process.

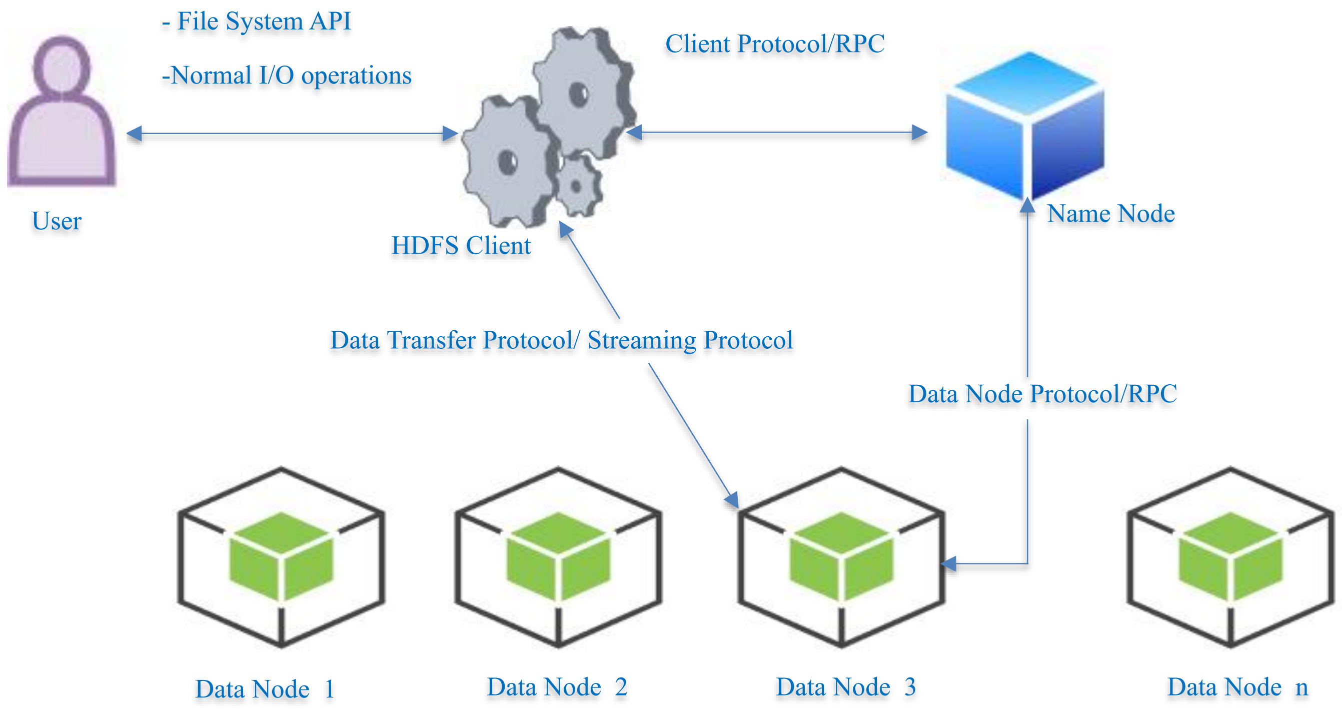

Fig. 2 Communication protocoles involved in HDFS read operation

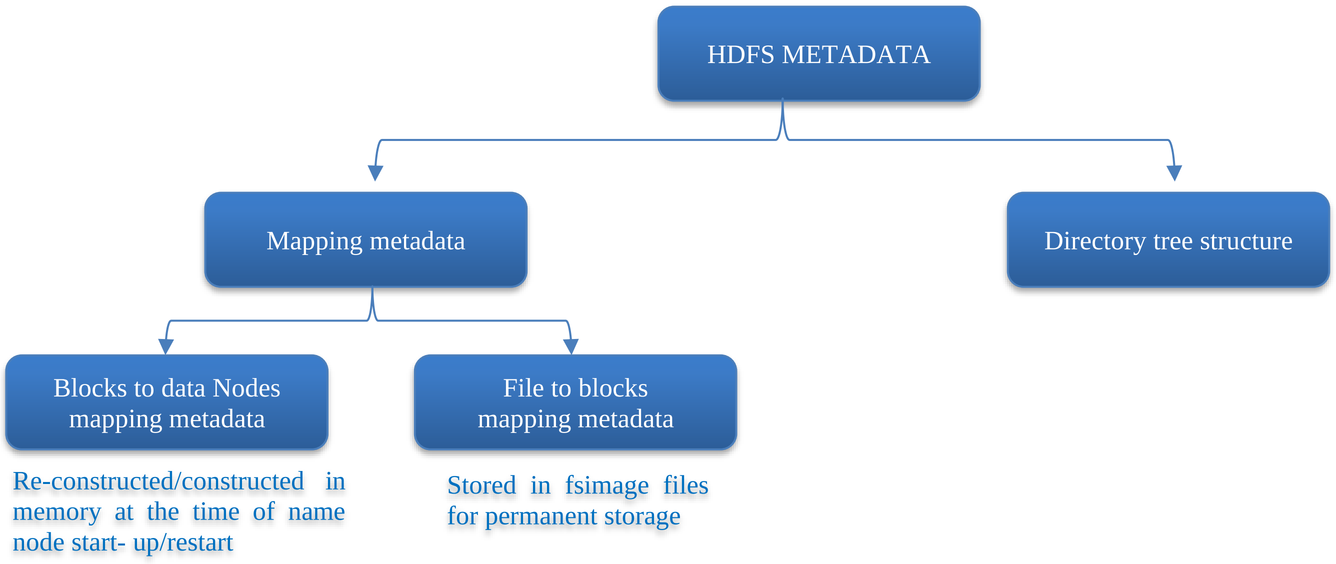
HDFS METADATA
Mapping metadata
Directory tree structure
Blocks to data Nodes mapping metadata
File to blocks mapping metadata
Re-constructed/constructed in memory at the time of name node start- up/restart
Stored in fsimage files for permanent storage

Fig. 3 HDFS MetaData categories and role

3.1. Role of HDFS MetaData for Reading

HDFS MetaData contains (Figure 3):

- The structure of HDFS directories and files in a tree.
- The relation between file and block.
- the relationship between block and DataNode

It also encompasses various attributes of directories and files, including ownership, permissions, quotas, and replication factors [16]. The NameNode constructs and maintains the latest metadata from block reports sent by DataNodes, providing an up-to-date view of block locations within the cluster [17]. In HDFS, everything is an object, whether file, block, or directory and it is represented by approximately 150 bytes per object in the NameNode MetaData. The entire MetaData is loaded into the memory of the NameNode for its operation. Thus, the memory of the NameNode is a limiting factor in the size of the cluster [18].

Mainly, there are two kinds of NameNode MetaData (Figure 3):

File System MetaData: It contains all the information about files (permissions for each file, timestamp, size, replication factor, and data blocks of that file). File system MetaData is of two categories:

FsImage: The FsImage is stored as a file in the NameNode's local file system. It contains the entire file system namespace, including the mapping of blocks to files and file system properties. The FsImage represents a snapshot of the file system that the NameNode uses upon startup, including the very first start. This snapshot is updated periodically to reflect changes in the file system. The FsImage always persists to disk, ensuring that data is not lost even if the NameNode shuts down.

Journal: or edit Logs continuously record every change that occurs to the file system, such as writing, deleting, updating, permission changing, or other types of transactions, and store them in an edit log that keeps on growing (edits inprogress file). When edits in the progress file reach a certain amount of transactions, or after a certain period of time, it will be renamed as an edit file. All the generated edit files will be merged with the existing FsImage to create the new FsImage either after NameNode rebooting or during checkpoints. A checkpoint can be triggered at a given time interval or after a given number of filesystem transactions have accumulated [13, 19].

In-memory MetaData: At the NameNode start, it reads the FsImage from the disk and loads it to memory. After that, all the transactions performed by the NameNode are added simultaneously to the FsImage in memory and to the edits_inprogress file in the disk. After a checkpoint, the content of In-memory FsImage will be equal to FsImage updated by all the edits files and the current edits inprogress file [13, 19], as illustrated in Figure 4.

NameNode loads MetaData in "in-memory" in order to serve the multiple Client requests as fast as possible. The secondary NameNode periodically does a checkpoint to update the FSImage so that the NameNode recovery is faster.

3.2. Description of a File Reading Process in HDFS

To read a file located in HDFS, the HDFS Client, on behalf of the user, communicates with the NameNode and the DataNodes. Users do not have direct access to the DataNodes where the actual data is stored according to its privileges; NameNode responds by returning a list of relevant DataNodes where the data resides [12] Read operations as write operations in HDFS occur at a block level. The DataNodes maintain continuous communication with the NameNode to determine if they need to complete specific tasks. This ensures that the NameNode is always aware of the status of the DataNodes. When a DataNode appears not to accomplish its tasks properly, the NameNode assigns them to another DataNode containing the same data blocks. User establishes communication with HDFS through File System API and normal I/O operations. Therefore, processing of user request and providing response over it is carried out by File System API processes.

The scheme in Figure 5 presents the different steps followed by a user to read a file located in HDFS.

The following subsections detail:

- The security issues related to HDFS read operation
- The HDFS Client interaction with the NameNode.
- HDFS Client interaction with the DataNodes.

3.2.1. HDFS Client Interaction with NameNode

Step 1: The user establishes communication with the HDFS Client and asks to read a file.

Step 2: The HDFS client initiates the opening of the file the user wishes to read by calling the open() method of the FileSystem object, which is an instance of the DistributedFileSystem (DFS). This triggers a read request to access the desired file.

Step 3: By using Remote Procedure Calls (RPC), the FileSystem object connects to the NameNode processes to determine the locations of the first few blocks of the file.

Step 4: The NameNode checks first if the user has sufficient privileges to get the file. If not, the user's read request is aborted.

Step 5: When the user has sufficient privileges, NameNode processes get MetaData information such as the locations where the replica is made, the number of DataNodes, the number of data blocks and their location from the In-memory FSimage.

Step 6: After gathering the necessary MetaData information and in response to the received MetaData request, the NameNode returns back the flow of MetaData that includes the addresses of the DataNodes having a copy of the file blocks. Moreover, the list of DataN odes is sorted based on their proximity to the Client [12]. NameNode responds with a read.

3.2.2. HDFS Client interaction with DataNodes

If the Client is itself a DataNode, the Client will read from the local DataNode in case this later hosts a copy of the block [14].

Step 7: In the HDFS Client machine, The DistributedFileSystem returns an FSDataInputStream to the client process for reading data. The client then uses the FSDataInputStream object to identify the data blocks on the DataNodes. FSDataInputStream, in turn, wraps a DFSInputStream, which manages the interactions with DataNodes and NameNode and stores the DataNodes addresses for the first few blocks in the file.

Step 8: The client invokes the read() method, causing the DFSInputStream to establish a connection with the first DataNode containing the initial block of the file. The client presents the block access token provided by the NameNode to the DataNode. Upon receiving the request, the DataNode verifies the authentication information of the requested block [20]. Thus, a TokenAuthenticator is created using the received block access token ID and the already shared secret key. The created and received TokenAuthenticators are then compared. If they are the same, authentication is completed [21], and then the Client starts reading data from the DataNode.

Steps 9 and 10: Data is read in the form of streams wherein the Client invokes the 'read()' method repeatedly. This process of read() operation continues till it reaches the end of blocks. The data flows directly to the Client, who reads the data block from multiple slaves in a parallel manner. If the DFSInputStream encounters an error while communicating with a DataNode during reading, it will attempt to connect to another available DataNode that stores the same block.

Additionally, the DFSInputStream keeps track of DataNodes that have failed to avoid retrying them unnecessarily for subsequent blocks. It also verifies checksums for the data transferred from the DataNode. If any corrupt blocks are detected, the DFSInputStream reports this issue to the NameNode, which then takes appropriate action to read a replica of the block from another DataNode [12, 14].

The DataNode may be lost down due to network errors, hardware failure, or block corruption. Fortunately, HDFS maintains the availability of data through further copies of the blocks thanks to replication. DFSInputStream may also call the NameNode to retrieve the DataNode location for the next batch of blocks as needed.

Step 11: Once the end of a block is reached, the DFSInputStream closes the connection with the current DataNode and proceeds to locate the next DataNode to access the subsequent block.

Step 12: Once a Client has finished reading, it calls a close () method on FSDataInputStream. The Client reads the data in a parallel way since the replica of the same data is available on the cluster. Once all the data is read, all the blocks are combined to form the original file.

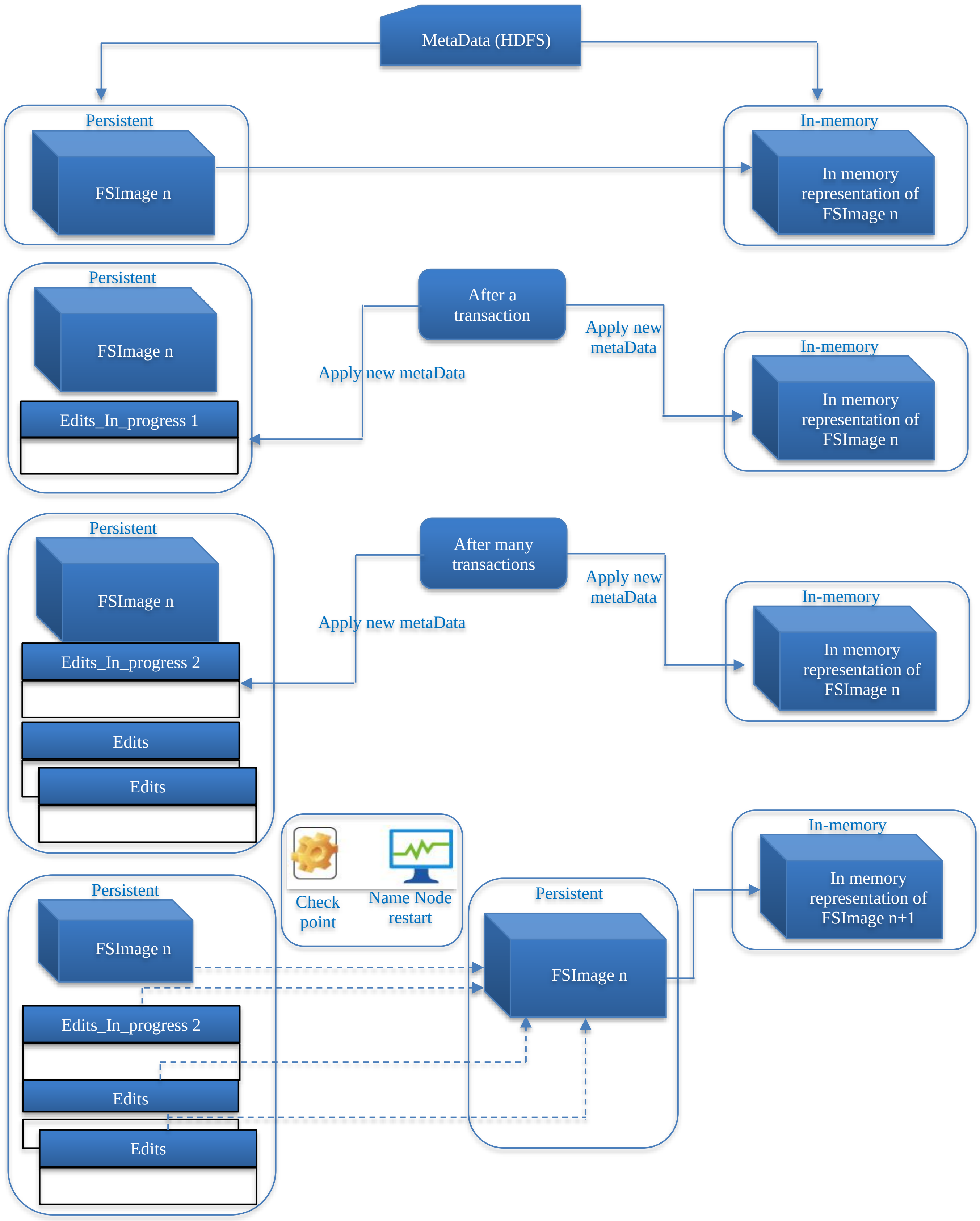

Fig. 4 NameNode metadata structure and evolution

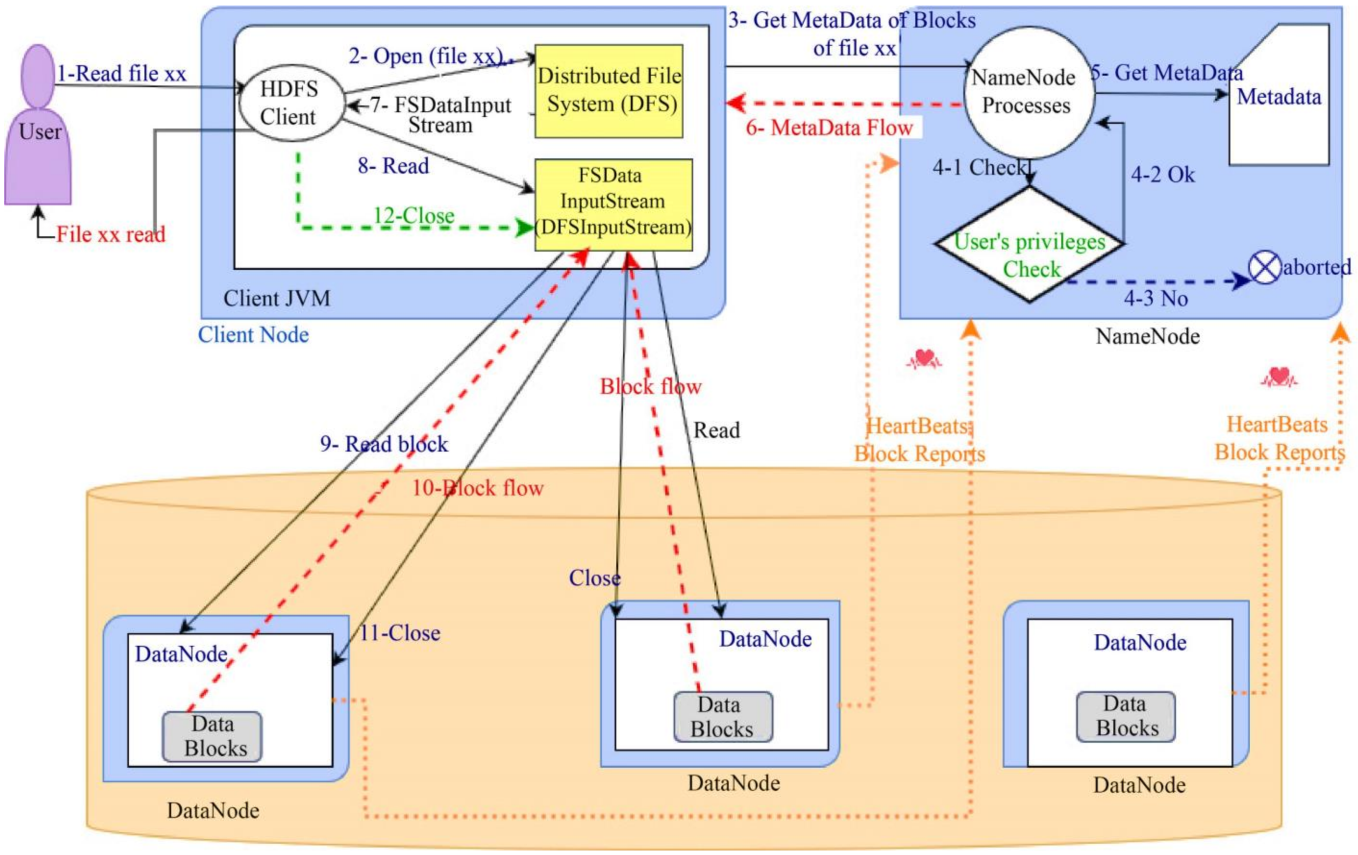

Fig. 5 Description of Read operation in HDFS

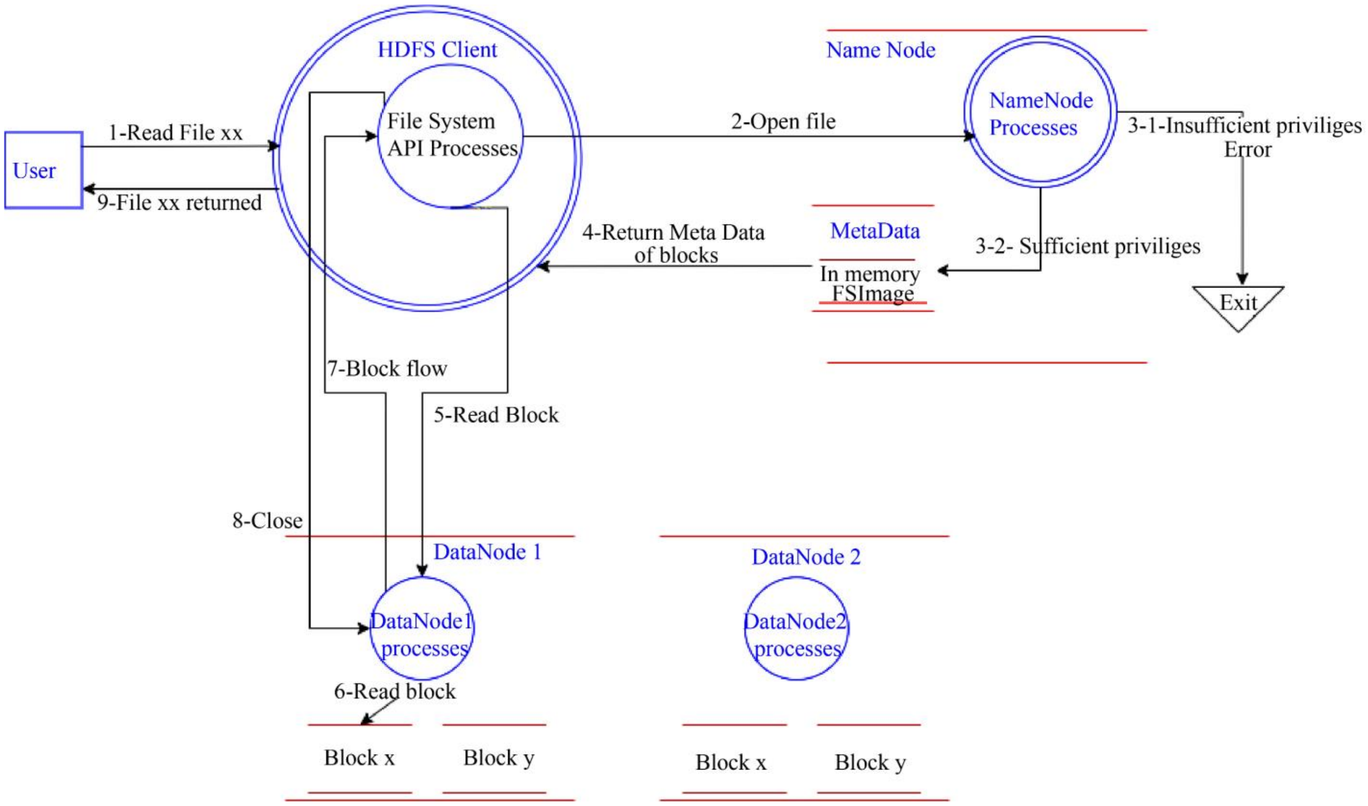

Fig. 6 Data flow diagram of HDFS Read Operation

4. HDFS Read Process Security Analysis

Although the Hadoop community supports several security features, such as Kerberos authentication, firewalls, data encryption, basic HDFS permissions, and Access Control Lists (ACLs), security is still not fully comprehensive. These measures have limitations that may enable attacks on assets stored in a data store in one of the components involved or along the network connecting the communicating parties. This section begins with an analysis of the architectural design of the HDFS read operation through a Data Flow Diagram (DFD). This step allows a clear definition of the boundaries and attack vectors. A second step identified the principal assets of the HDFS read operation and their associated security requirements.

4.1. Determination of Security Scope

The present subsection proposes a Data Flow Diagram (DFD) that will be the basis of the HDFS read process security use cases study. A Data Flow Diagram (DFD) provides a graphical representation of the data flow within an information system. It helps in understanding processes and applications by illustrating how data moves through the system. The DFD includes both internal elements (such as processes and data stores) and external elements (such as external entities), all connected through communication channels that depict the flow of data.

These elements include 'Processes', 'Data Stores', 'Data Flows', and 'External entities', and work reasonably well as a means of eliciting information that can be used to drive analysis [22]. In this way, a DFD creates the opportunity to identify where (important) data is coming from and how it is processed and stored, enabling easier identification of security risks, vulnerabilities, and threats. Problems tend to follow the data flow [23]. Generally, such problems tend to cluster around trust boundaries that delineate the attack surface [23]. Figure 6 illustrates the Data Flow Diagram of the current study and shows the interaction between different processes and stores:

Processes include File System API processes, NameNode process, and DataNode processes; Stores include NameNode store, MetaData sores, DataNode stores, Data blocks stores, and In memory FSImage store;

External entities may be external users or an external program that wants to read files from HDFS. We have identified, as shown in Figure 6, nine flows:

Flow 1: Between the external user and the HDFS Client is a request to read a file from HDFS.

Flow 2: Between the File System API Processes that belong to the HDFS Client and the NameNode Process: This flow is about asking the NameNode process for the file opening;

Flow 3: This flow occurs inside the NameNode and checks the user's ability to read the requested file. It can be one of two subflows:

Sub-Flow 3-1: The NameNode judges that the user does not have sufficient privileges to read the file, then it terminates the read request without any result;

Sub-Flow 3-2: The NameNode judges that the user has the necessary privileges to read the file and contacts the MetaData Store to return back the necessary MetaData;

Flow 4: MetaData of the datablocks that constitute the requested file is extracted from the FSImage and then sent to HDFS Client. The MetaData includes the addresses and the locations of the data blocks;

Flow 5: According to the received data blocks, the HDFS Client processes contacts the nearest DataNode that contains the first data block;

Flow 6: The DataNode Process gets the requested data Block;

Flow 7: The DataNode Process sends the Data Block to the File system API Processes that belong to the HDFS Client;

Flow 8: The HDFS Client closes the connection with the DataNode and moves on to the next DataNodes that contain the following data blocks.

Flow 9: When the HDFS Client receives all the data blocks that constitute the requested file, it returns this later to the user.

4.2. HDFS Read Process Security Requirements

Assets include data, data stores, permissions and privileges, functions, applications, and network exchanges. The assets involved in the information flow should be defined, and their security requirements evaluated based on their importance in the process.

The security requirements include confidentiality (sensitive data should not be accessible to attackers or unauthorized persons), integrity (attackers should not modify process operations or data stores), availability (data should remain accessible even under attack), and authorization (an actor must have the necessary permissions and privileges to access an asset, such as reading a file stored in a DataNode). For this study this study classifies assets of the HDFS read process into three categories; many security issues and requirements may concern one category:

- Data, MetaData, key, token and stores
- Network exchanges and flows
- System processes.

4.2.1. Data, MetaData, Key, Token and Stores Security Issues

Data and data stores are the most valuable assets that attract attackers' attention. These latter could target the confidentiality, integrity and availability of data either stored permanently in data stores, like DataNodes or temporarily, in the memory, such as In-memory MetaData, available in the NameNode memory during its operation. Indeed, in-memory MetaData contains the blocks mapping between the files existing in HDFS and the file system properties.

In addition, MetaData is updated with new information after every transaction done by the NameNode. Thus, the availability of these MetaData for the HDFS Client process is vital to accomplish a file reading. Data Blocks are small units of data located in DataNodes and constitute the data storage in HDFS together. They must be available and accessible only to authorized users or processes. They also need to be protected against alteration by unauthorized users or processes. Initially, the Hadoop system permits access to a datablock located in a DataNode with no control [20].

The idea of a block access token was introduced with Hadoop 1.0.0 to confirm access authorization to the data block. Since it is used as the authentication mechanism, it must be protected against confidentiality and integrity attacks and must be held by authorized entities. On the other hand, each actor who wants to access HDFS for reading purposes must authenticate. The system then applies authorization rules according to the access control rules already defined. The HDFS implements a permission model for files that shares much of the POSIX model [22]. There are generally three permissions for a file: READ, WRITE, and EXECUTE. To read a file in HDFS, the" R" permission is required [22]. Keeping file permissions, ownership, and user privilege preserved implies that confidentiality and integrity must be totally assured and that unauthorized actors or processes should not alter permissions.

4.2.2. Network Exchanges Security Issues

Network and flow security issues encompass threats that impede the normal flow of communications and data through the network. These threats mainly affect confidentiality and integrity and are generally related to man-in-the-middle attacks. As shown in Figure 6, through the network, a user remotely requires reading a file from HDFS, and the request lands into the HDFS Client. The HDFS Client remotely communicates with the NameNode to get the MetaData of the Data blocks that constitute the requested file to be read. There are also remote communications between the NameNode and the DataNodes through heartbeats, as well as between the DataNodes and the HDFS Client. All these network exchanges are mainly concerned with confidentiality and integrity issues.

- MetaData in Transit: The confidentiality and the integrity of the MetaData that flow between the NameNode and the HDFS Client must be secured against disclosure or alteration by unauthorized actors.
- Data blocks in transit: When flowing one by one from DataNodes to the HDFS Client, their contents will constitute the final file that is requested. Thus, the confidentiality and integrity of each data block in transit must be preserved.
- Block access token in transit: The block access token serves as the client's authenticator to the DataNode. When flowing from the NameNode to the HDFS Client or from this later to a DataNode, the block access token must be secured against divulgation or alteration. Taking control of a block access token or altering the related token authenticators might prevent the reception of the requested datablock and thus prevent the file from being read or at least delay its reading.
- File in transit: When the whole requested file is constituted from the necessary data blocks, the HDFS Client sends it to the end user. Thus, the resulting file from the whole read operation has to be protected against divulgation or alteration by unauthorized attackers and its confidentiality and integrity must be preserved.
- Secret key in transit: The secret key is a very sensitive credential in the HDFS read operation and must be secured when transferred from the NameNode to the DataNode at each update.

It is frequently updated between the NameNode and a DataNode and permits the creation and encryption of the block access token [20]. Intercepting the transferred secret key implies that data blocks residing in the DataNode are exposed to attacks.

4.2.3. System Processes Security Issues

Security issues related to the system processes rather concern the availability of a process for the tasks it is attended to and also its ability to perform the tasks that are essential to read a file from HDFS.

The essential tasks that could be subject to attacks include the ability of the NameNode processes to check the user privileges and its eligibility to read the requested file, and also their ability to correctly retrieve the necessary MetaData from In-memory FSImage and to sort them according to proximity between DataNode and the Client. Concerning the DataNode processes, their availability is crucial to properly processing the request for the dataBlock. Thus, an erroneous DataBlock or no DataBlock returned to the HDFS Client caused by a denial-of-service attack, which could delay the duration of the read operation. The HDFS Client processes seem to be the busiest processes in the operation of reading the file from HDFS. The related security issues include hindering them from submitting the opening file request to the NameNode or preventing them from reconstituting the final file and verifying its integrity through the checksum. The HFDS client processes handle multi-read requests and could be subject to attacks that reduce this ability by preventing them from satisfying multiple read requests.

4.3. Existing Security Solutions

Initially, the Hadoop framework was developed without integrating security aspects [1, 24]. Then, there were many attempts to add security to this environment. Indeed, the proposed existing security solutions were generally built upon encryption methods [1, 24, 25, 26] to preserve the integrity and confidentiality [24, 25] of HDFS files. However, most encryption techniques result in increasing the size of the files and compromising the system's performance at different levels [1, 25]. In a Hadoop environment, all users and services initially had the same level of access privileges to all data in the cluster, which posed security risks [26]. This lack of granular access control made it easy for any user to impersonate others, potentially leading to unauthorized access or misuse of data [24]. In this context, no authentication of users or services was required, and data privacy was not protected. Later, the authorization and authentication aspects were added without being effective enough, given the numerous weaknesses of the security system of the Hadoop ecosystem [24]. Over the past ten years, twenty-nine Hadoop vulnerabilities and security weaknesses have been reported and fixed, of which 20 vulnerabilities have a Common Vulnerability Scoring System (CVSS) score greater than 5 [9, 22], which means severe vulnerabilities. On the other hand, the work [20] sheds light on the block access token to address the related permission control issues. It proposes a secret sharing-based block access token management scheme to overcome the security vulnerabilities targeting the block access token.

In Fact, when a user accesses HDFS to read a file, the HDFS Client contacts the NameNode to get data block locations, and then the HDFS Client accesses blocks directly on the DataNodes. Permissions checks by users upon a file are done at the level of the NameNode. To access the data blocks access in the DataNodes, authorization is required and assured by block access tokens, which are sent by the NameNode to the Client and then passed by this later to the concerned DataNodes. The concept of a block access token was introduced to confirm the access authorization to the data block. It is composed of a token ID and a TokenAuthenticator. A single block is sent with a single block access token to represent the authentication information of the block [20].

For Hadoop security purposes, the NameNode and the DataNode share a secret key used to calculate the TokenAuthenticator of a block access token. It is randomly chosen by the NameNode and sent to a DataNode when it is first registered with the NameNode. It is updated by the NameNode periodically and shared with the DataNode via heartbeats. The block access tokens are sent after being encrypted using the secret key upon a user's request [20]. It is noteworthy that when the secret key is transferred between the NameNode and the DataNode, it is exposed to attackers and the blocks stored in the DataNode are consequently exposed to attacks. Various security threats could also happen to the block access tokens [20]. Thus, securing access to the secret key and blocking access tokens are crucial for the security of the HDFS read operation. Concerning HDFS as a component of the Hadoop system, it shows a remarkable lack of security preservation, and the Hadoop framework is considered not mature enough to deal with cyber-attacks on data [1]. Its security is threatened since it holds one of the most valuable assets of an organization, which is the Data, and without appropriate and permanent security measures, unauthorized data access, data theft, and unwanted disclosure of information could happen, which would affect consumer trust in the system provider [27]. Thus, troubles might arise, and the consequences could be devastating and unrecoverable. Studying and analyzing security threats is the first step towards taking precautions and establishing protection boundaries. Some of the threats are mature and hard to tackle, while some are of a lower level and easy to handle [28]. This would be the main objective of the following section.

5. Security use Cases Threat Analysis and Mitigation Study

The following subsections focus on some crucial security use cases related to different types of HDFS Read operations assets. Their threats will be analyzed, and some mitigation solutions will be discussed.

5.1. Use Cases

After reviewing various security issues related to the operation of reading files from HDFS in the previous section and showing their impact on the smooth running of a file reading process, this paper meticulously selects the main 3 use cases of attacks. The objective is to show their critical impact that may abuse the HDFS read operation.

5.1.1. Use Case 1: MetaData Targeted by Tampering, Information Disclosure or Denial of Service Attacks

The first use cases category is about MetaData security in the NameNode machine. The latter stores file system metadata in two different files: the FSimage and the editlog. The FSimage stores a complete snapshot of file system metadata at a given point in time. All subsequent changes are stored in editlog and transferred to FSimage under certain conditions in order to restore the file system to its most recent state [29]. An attacker could gain access to the NameNode machine, and then two security issues could happen: The attacker could access the read or write memory inappropriately through a successful stack or heap buffer overflow. An example of this type of vulnerability is CVE-2021-37404 [30], affecting some Apache Hadoop versions. It consists of a heap buffer overflow in the libhdfs native library, which could cause a denial of service or arbitrary code execution. This kind of vulnerability could lead to alteration (Tampering attack) or suppression of some entries (Denial of service attack) of editlog located in the memory and then loss of important information about files data stored in the HDFS [30]. On the other hand, the attacker, instead of accessing the MetaData in memory, could target the

FSimage stored in the disk. He could exchange FSimage with an older one or a different one (Tampering attack). A vulnerability like CVE-2018-11768 [30] could lead to FSimage corruption and even more to information disclosure since it causes the user/group information to be corrupted across storing in FSimage and reading back from it.

5.1.2. Use Case 2: Privilege Escalation and User Impersonation.

Privilege escalation is a common vulnerability detected in many Apache Hadoop versions at different levels (WebHDFS, Web endpoint, YARN framework). The following already detected and documented vulnerabilities testify to this: CVE-2021-33036, CVE2020-9492, CVE-2018-11764, CVE-2018-8029, etc [30]. In many versions, these vulnerabilities could lead to serious damage. A broken authentication check, for example, could enable some simple users to impersonate a privileged user and even the root that has many more administrative privileges. This would allow the attacker to perform many malicious tasks on the system:

- Modifying stored data;
- Deleting important files;
- Disabling or removing active processes and then replacing them with malicious ones;
- The attacker could even install some trojan or backdoors to be able to perform remote control of the HDFS components.

5.1.3. Use case 3: Session Hijacking and MiTM Attacks

The network is full of various exchanges between remotely interconnected processes and stores. Thus, attackers could use many attack vectors to break these communications. These attacks include Attacks that disturb the normal transfer of MetaData between the NameNode and the HDFS Client process by intercepting, altering or/and redirecting the flow of MetaData to the attackers' machine instead of the HDFS Client process. Here are some examples of cases that fall into this category:

– An attacker may perform session hijacking by modifying the MetaData of the file that flows over the network, especially bymodifying data block addresses or locations;
– The file path in transit (DFD-Flow 2): Modifying the requested file path in the HDFS Client open request sent from the Client to the NameNode (Man-in-the-middle attack) in order to read the file needed by the attacker instead of the onerequested by the user;
– MetaData in transit (DFD-Flow 4): Redirecting the flow of MetaData to theattacker's machine;
– Preventing the sending of the locations of the following blocks, which cause the file not to be read;
– Other important security problems could happen through the network during the exchanges between the HDFS Client and the DataNodes to retrieve the Data blocks and could lead to information disclosure, denial of service, or even tampering attacks:
– The attacker could intercept the block access token in order to illicitly read the data blocks or create a fake access token able to intercept the secret key exchanged between NameNode and DataNode in order to be able to read the data block from the DataNode.
– The attacker could redirect the flow of data to his machine in order to prevent the HDFS Client from accessing its content and retrieving data blocks or by modifying data blocks flowing over the network.
– Modifying the checksum flowing through the network by the attacker may make the HDFS Client process unable
– to validate the received data blocks. In this way, the file could not be read by the end user.

5.2. Discussion Threat Analysis and Mitigation Proposition

Having examined the impact of the main use cases on the read operation in HDFS in the previous section, the current section examines the various threats relating to these use cases, categorizes them, and then proposes some possible solutions for mitigation.

5.2.1. Threats Analysis

The technical impact of different threats is considered by mapping them in STRIDE Threats categorization. STRIDE, which is a Threat Modeling method, stands for" Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation" which are the opposites of the principal security requirements: Authentication, Integrity, Non-Repudiation, Confidentiality, Availability and Authorization. In fact, one efficient solution to enhance a system's security is threat modeling, a widespread approach in system security evaluation [7] that helps give a view of the possible threats that may affect a system's security or its proper functioning. By performing this threat analysis, this paper is making a preliminary threat modeling of the HDFS read process to have a clear idea of the security impact of the detected threats and then study the possible solutions to mitigate them. Table 1 addresses the main threats related to the use cases previously developed.

Table 1 provides a mapping between technical impacts and their associated threats, including Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privileges. It illustrates how one technical impact can result from multiple threats. For example, an unavailability of MetaData (Editlog or Fsimage) may be triggered by tampering and denial of service threats. The use cases examined above show that HDFS file reading operations are mainly affected by security threats linked to tampering, information disclosure and denial of service. So, it seems that these types of threats need to be anticipated as much as possible in order to control and mitigate them and reduce their impact.

Table 1. Impact of the studied uses cases

Use Case	Technical Impacts	Threat categories: Spoofing	Tampering	Repudiation	Info. Disclosure	Denial of Service	Elev. of Privileges
Use Case 1	Loss or downtime of the MetaData that hinders the HDFS Client from getting the data block addresses and locations					X	
	Unavailability of some entries of editlog located in the NameNode memory (after Alteration or deletion), which may result in denial of service to the NameNode information		X			X	
	Unavailability of the current FSimage (after deletion or exchange by an older one or a different one)		X			X	
Use Case 2	Impersonation attack to accomplish actions on data, data stores, file permissions and MetaData	X	X		X		
	Unavailability of necessary active processes (after removing or replacement by malicious ones)		X			X	
	Illegal remote control of HDFS components (by installing trojans or backdoors) or execution of malicious tasks in NameNode, DataNodes or HDFS Client		X	X			X
Use Case 3	Reading a different file, rather than the requested one, by carrying out a man-in-the-middle attack on the request to open the file required by the user		X		X		
	Unavailability of the requested MetaData for HDFS Client processes (MetaData flow is redirected to the attacker’s machine		X		X	X	
	Unavailability of the following data block locations, which will prevent the user from reading the file					X	
	Illicit access to data block either in DataNode (by stealing or forging a fake block access token) or through the network (by conducting a successful MiTM attack) to prevent the HDFS Client from receiving it	X	X		X	X	

5.2.2. Discussion on Possible Mitigation Techniques Related to the Use Cases

Table 1 shows the main threats encountered when reading a file stored in HDFS. The following paragraphs discuss the mitigation of detected threats:

- Mitigating threats related to Spoofing: Spoofing-related threats target authentication and consist of the Impersonation of a user to accomplish actions on data, data stores, file permissions and on MetaData or to install.

Trojans and backdoors to remotely control HDFS or even perform MiTM attacks to read other files. All these threats can be accomplished during a session spoofing or Session Hijacking. In order to avoid these latter attacks and to ensure the authentication of the communicating parties, the use of a secured communication protocol is very crucial. Kerberos, the system for authenticating access to distributed services, is already used as an authentication mechanism in Hadoop. It ensures authentication using limited-time tickets/tokens that could limit unauthorized access in case they are stolen. However, Kerberos doesn’t address data encryption. So, since the communication channels between services are not secure, then the credentials could be intercepted or a new communication forged. Another important issue is that time needs to be consistent across all involved machines to ensure that the time-limited tokens are working.

For these reasons, it would be interesting to use a secured protocol like HTTPS (Hyper Text Transfer Protocol Secure) based on TLS (Transport Layer Security) or Secure Shell (SSH) as a communication protocol between different components. It ensures, in addition to authentication, data integrity and confidentiality. In this context, Virtual Private Network (VPN) seems to be an option but it would be heavy to use inside the Hadoop Architecture. In the opposite to VPN, it would be promising to explore the possibility of using Zero Trust Nework Access (ZTNA), the new evolving approach in the network security field. It has the advantage of offering control over access and movement over the network without granting relatively open access to resources and equipment using a single authentication. Indeed, Zero Trust allows authentication and receiving a set of permissions and authorization for explicit access [31].

- Mitigating threats related to Tampering: Tampering threats compromise data integrity through the modification or deletion of either data in rest stored in editlogs, FSimages, datablocks or in motion by altering data transferred across the network. In addition, the alteration could also impact the processes by replacing a legitimate process with a malicious one, users or group information or the modification of the file path in the read open request.

These threats could be mitigated through the reinforcement of the Access Control Lists (ACL) mechanisms. In this way, ACLs could be applied at the sides of HDFS Processes, the NameNode, and each DataNodes. The tampering threats may also be mitigated through the implementation of digital signatures. A digital signature guarantees that authenticated users or processes send the exchanged messages, files, and data requested and are not altered in transit. This could constitute an efficient security measure for credentials, such as block access tokens that circulate through the network. Furthermore, as mentioned before, the use of a secured communication protocol between nodes, like HTTPS, assures data integrity and correct protection against MiTM attacks.

- Mitigating Repudiation Threats: Processes are the main victims of repudiation attacks. These attacks involve pretending not to have performed an action or to have been the victim of fraud or attacking logs by deleting, modifying or accessing their contents. For the use cases, one repudiation threat is identified, and it concerns an illegal remote control of HDFS components (by installing trojans or backdoors) or execution of malicious tasks in NameNode, DataNodes, or HDFS Client processes.

Repudiation attacks could be mitigated through fraud prevention, log analysis and the use of digital signatures. It is worth noting, in this context, that traceability is very important to face these types of threats. So, protecting log files and audit events from alteration by malicious malwares, like Rootkits, is crucial to being able to consult each task user or process activity history in one specific node.

- Mitigating threats related to information disclosure: These threats mostly aim to access confidential data either stored in NameNode or DataNodes or transmitted through the network between the node exchanges. Using Encryption is the principal good solution to ensure confidentiality.

The actual versions of Hadoop already implement this mechanism in different levels: (i) Volume encryption level that protects data after physical theft or accidental loss of the disk. However, this mode of encryption does not provide a fine-grained encryption of some critical files or data stores and does not protect against malwares or other types of attacks when the system is running. (ii) Application-level encryption allows for better data protection. (iii) HDFS ”Data at Rest” encryption [32] that allows end-to-end encryption of data stored in the HDFS performed by the HDFS Client, which seems to be a good measure to protect data confidentiality at rest. Another important measure to mitigate confidentiality threats is to implement ACL mechanisms to protect directories, filenames, and file contents. In addition, for the protection of data and communication confidentiality through the network, the use of a secured protocol is a must, as explained above.

- Mitigating Denial of Service (DOS) threats: These threats compromise the availability of processes, data stores and data flows and can manifest themselves in the loss, deletion or alteration of MetaData in the NameNode or by the replacement of original processes by malicious ones in one of the principal nodes involved in the HDFS read operation (NameNode, HDFS Client or DataNodes).

The denial of service could also be performed by redirecting MetaData flow to processes other than the HDFS client process that is supposed to receive them. Furthermore, these types of threats can also concern the flow of data blocks to the HDFS client process. DoS attacks could threaten the nodes themselves and make them unable to perform their respective tasks. DoS mitigation is a process that aims to minimize the impact of these threats on the overall HDFS read process. On the one hand, in each involved node, a rate limitation either at the network level could be implemented to limit the number of accepted requests in a certain amount of time (in the NameNode, for example). On the other hand, it could be achieved by limiting the number of API calls since a repeated and large number of API calls could lead to the unavailability of the called process. This last issue could be very useful either in the NameNode/DataNode or in the HDFS Client. At the node level, the intrusion tolerance approach could be introduced in order to enhance the architecture tolerance for intrusions, especially DoS or DDoS (Distributed DoS) attacks and shorten or even avoid downtime in case of received attacks. Monitoring traffic through the use of Intrusion Detection Systems (IDS) and traffic filtering by using either network firewalls or host-based firewalls are other important protection measures against these types of threats.

- Mitigating Elevation of Privileges threats: This category of threats could be addressed when an attacker reads the file requested by the user or reads a different file by injecting another filename or file path into the user’s request. This threat could be mitigated by reinforcing input validation and reworking the granting of roles and access rights for group users in addition to file permissions.

Indeed, one of the most efficient measures, especially against lateral movement threats, is a good authentication policy. Choosing good and complex passwords for the existing user could render this type of threat more difficult. On the other hand, it is worth noting that the vertical escalation of privilege, where the attacker manages to gain access to a privileged user or even to the super-user, is usually performed by exploiting system misconfigurations, vulnerabilities (in system or application level) or porous access controls rules. This could be mitigated by a regular update of the software used in each node and good maintenance of the Java codes. In addition, implementing a fine-grained and appropriate access control would be an efficient mitigation measure against this type of threat.

6. Conclusion

This article is devoted to addressing the security threats associated with reading files stored on HDFS for two reasons: the first being that HDFS plays a very important role in both storing and managing large volumes of data and files while ensuring data availability. The second reason is that, while HDFS is in great demand for reading its files and data, it is not totally immune to security threats that can impede accessing and reading its files, as existing solutions remain insufficient to offer acceptable security of HDFS, particularly for the read operation. This lack of security, which would lead to difficulties in accessing the necessary files, would cause delays in making decisions and taking action in the shortest possible time and thus result in lost opportunities. To this end, through rich and varied illustrations and documentation, this paper has shed light and given a holistic description of how a file stored in HDFS is read, including the exchange of the relevant data blocks and MetaData. The different interactions between HDFS components, including DataNodes, the HDFS Client, the NameNode and associated stores and processes, are deeply examined. Through a detailed analysis of existing security solutions as well as the security issues and requirements of the HDFS file reading operation, this paper has shown the inadequacy of existing solutions to meet the related safety requirements. Three (3) main attack use cases that can adversely affect the HDFS file reading operation were thoroughly selected and discussed. Then, the impact of each use case on the involved assets is discussed by describing the threats related to each use case and mapping them into 6 categories of threats, including Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privileges. Finally, mitigation measures capable of reducing their impact are suggested for each category of threats; they help guarantee better security for reading the files stored in HDFS. Future work could include similar studies to identify security threats related to other operations performed on HDFS, with the aim of mastering all security threats to HDFS and related operations.

References

[1] Gurjit Singh Bhathal, and Amardeep Singh Dhiman, "Big Data Security Challenges and Solution of Distributed Computing in Hadoop Environment: A Security Framework," *Recent Advances in Computer Science and Communications*, vol. 13, no. 4, pp. 790-797, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[2] Boris Lublinsky, Kevin T. Smith, and Alexey Yakubovich, *Professional Hadoop Solutions*, John Wiley and Sons, pp. 1-477, 2013. [Google Scholar] [Publisher Link]

[3] Mark Grover, *Hadoop Application Architectures: Designing Real-World Big Data Applications*, O'Reilly Media, pp. 1-400, 2015. [Google Scholar] [Publisher Link]

[4] Vladimir Kaplarevic, Apache Hadoop Architecture Explained (with Diagrams), Phoenixnap Global IT Services, 2020. [Online]. Available: https://phoenixnap.com/kb/apache-hadoop-architecture-explained

[5] Karwan Jameel Merceedi, and Nareen Abdulla Sabry, "A Comprehensive Survey for Hadoop Distributed File System," *Asian Journal of Computer Science and Information Technology*, vol. 11, no. 2, pp. 46-57, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] Yordan Kalmukov et al., "Analysis and Experimental Study of HDFS Performance," *TEM Journal*, vol. 10, no. 2, pp. 806-814, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Xiong Wenjun, and Robert Lagerström, "Threat Modeling – A Systematic Literature Review," *Computers & Security*, vol. 84, pp. 53-69, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[8] Kinza Yasar, Hadoop-Distributed-File-System (HDFS), [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/Hadoop-Distributed-File-System-HDFS

[9] Apache Hadoop, Apache Website, 2023. [Online]. Available: https://hadoop.apache.org/

[10] Anatomy of File Read and Writes in HDFS, Geeksforgeeks, 2022. [Online]. Available: https://www.geeksforgeeks.org/anatomy-of-file-read-and-write-in-hdfs/

[11] Subhi R.M. Zeebaree et al., "*Characteristics and Analysis of Hadoop Distributed Systems*," Technology Reports of Kansai University, vol. 62, no. 4, pp. 1555-1564, 2020. [Google Scholar]

[12] How Hadoop Works Internally – Inside Hadoop, Data Flair, 2023. [Online]. Available: https://data-flair.training/blogs/how-hadoop-works-internally/

[13] What do you Mean by Metadata in Hadoop HDFS?, Data Flair. [Online]. Available: https://data-flair.training/forums/topic/what-do-you-mean-by-metadata-in-hadoop-hdfs/

[14] Tom White, *Hadoop: The Definitive Guide*, O'Reilly Media, pp. 1-688, 2012. [Google Scholar] [Publisher Link]

[15] Hadoop Distributed File System (HDFS) Architectural Documentation, Hawaii, [Online]. Available: http://itm-vm.shidler.hawaii.edu/HDFS/ArchDocCommunication.html

[16] HDFS Administration, Backing up HDFS MetaData, Cloudera Documentation, 2023. [Online]. Available: https://docs.cloudera.com/?tab=cdp-public-cloud

[17] Yoon-Su Jeong, and Yong-Tae Kim, "A Token-Based Authentication Security Scheme for Hadoop Distributed File System using Elliptic Curve Cryptography," *Journal of Computer Virology and Hacking Techniques*, vol. 11, pp. 137-142, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[18] Gurmukh Singh, *Hadoop 2. x Administration Cookbook*, Packt Publishing, pp. 1-348, 2017. [Google Scholar] [Publisher Link]

[19] Ben Spivey, and Joey Echeverria, *Hadoop Security: Protecting your Big Data Platform*, O'Reilly Media, pp. 1-340, 2015. [Google Scholar] [Publisher Link]

[20] Su-Hyun Kim, and Im-Yeong Lee, "Block Access Token Renewal Scheme Based on Secret Sharing in Apache Hadoop," *Entropy*, vol. 16, no. 8, pp. 4185-4198, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[21] "*Securing Big Data: Security Recommendations for Hadoop and NoSQL Environments*," Securosis, L.L.C, pp. 1-18, 2012. [Google Scholar] [Publisher Link]

[22] Security-Vulnerabilities, cvedetails, 2023. [Online]. Available: https://www.cvedetails.com/vulnerability-list/

[23] Adam Shostack, *Threat Modeling: Designing for Security*, Wiley, pp. 1-624, 2014. [Google Scholar] [Publisher Link]

[24] Priya P. Sharma, and Chandrakant P. Navdeti, "Securing Big Data Hadoop: A Review of Security Issues, Threats and Solution," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 2, pp. 2126-2131, 2014. [Google Scholar] [Publisher Link]

[25] Hadeer Mahmoud, Abdelfatah Hegazy, and Mohamed H. Khafagy, "An Approach for Big Data Security Based on Hadoop Distributed File System," *2018 International Conference on Innovative Trends in Computer Engineering (ITCE)*, Aswan, Egypt, pp. 109-114, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[26] Mike Ferguson, "Enterprise Information Protection - the Impact of Big Data," *Intelligent Business Strategies*, 2014. [Google Scholar] [Publisher Link]

[27] Nataliya Shevchenko et al., "Threat Modeling: A Summary of Available Methods," *Software Engineering Institute*, pp. 1-26, 2018. [Google Scholar] [Publisher Link]

[28] Maahir Ur Rahman Mohamed Shibly, and Borja Garcia de Soto, "Threat Modeling in Construction: An Example of a 3D Concrete Printing System," *37th International Symposium on Automation and Robotics in Construction*, Kitakyushu, Japan, pp. 625-632, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[29] Hadoop-Architecture-Overview, Datadog, 2020. [Online]. Available: https://www.datadoghq.com/blog/hadoop-architecture-overview/

[30] Hadoop CVE List, Hadoop apache. [Online]. Available: https://hadoop.apache.org/cve_list.html

[31] Tim Keary, A Guide to Spoofing Attacks and How to prevent them in 2024, Comparitech, 2023. [Online]. Available: https://www.comparitech.com/net-admin/spoofing-attacks-guide/

[32] Encrypting Data at Rest, Cloudera documents, [Online]. Available: https://docs.cloudera.com/cdp-private-cloud-base/7.1.8/security-encrypting-data-at-rest/topics/cm-security-encryption-planning.html