A RAM Triage Methodology for Hadoop HDFS Forensics

Petra Leimich[a*], Josh Harrison[b], William J Buchanan[a]

[a] Centre for Distributed Computing, Networks, and Security, Edinburgh Napier University, Edinburgh EH10 5DT, UK

[b] Corax Cyber Security, 535 Mission Street, San Francisco, CA 94105, USA

Abstract

This paper discusses the challenges of performing a forensic investigation against a multi-node Hadoop cluster and proposes a methodology for examiners to use in such situations. The procedure's aim of minimising disruption to the data centre during the acquisition process is achieved through the use of RAM forensics. This affords initial cluster reconnaissance which in turn facilitates targeted data acquisition on the identified DataNodes. To evaluate the methodology's feasibility, a small Hadoop Distributed File System (HDFS) was configured, and forensic artefacts simulated upon it by deleting data originally stored in in the cluster. RAM acquisition and analysis was then performed on the NameNode in order to test the validity of the suggested methodology. The results are cautiously positive in establishing that RAM analysis of the NameNode can be used to pinpoint the data blocks affected by the attack, allowing a targeted approach to the acquisition of data from the DataNodes, provided that the physical locations can be determined. A full forensic analysis of the DataNodes was beyond the scope of this project.

Keywords

Digital forensics, Distributed filesystem forensics, Cloud storage forensics, Hadoop forensics, Triage, RAM forensics, Big data

1. Introduction

We are at an 'evolutionary point in a new era of the computing environment' (Daryabar et al, 2013). To satisfy the ever-increasing throughput requirements of big data, the use of distributed computing architectures is growing exponentially, and corporate giants such as Facebook, Amazon, and Yahoo! all now use data centres with thousands of nodes holding many petabytes of data. Such data stores pose an attractive target to criminals, and The Cloud Security Alliance (2014) name 'Data breach' and 'Data loss' as two primary threats to cloud computing, while the $(ISC)^2$ White Paper IX (2013) identified seven main concerns in relation to cloud security that include data loss, forensic readiness, and uninterrupted availability. It is clear from these concerns that despite the widespread adoption of distributed and cloud computing, there is uncertainty as to whether the technology can handle a data breach scenario. Garfinkel (2010) had already foreseen the coming challenges, with the bleak prognosis that digital forensics was at the end of its golden age, entering a time of crisis due to expanding technologies and their technical, business-centric and legal challenges.

Apache Hadoop is among the most implemented distributed computer architectures for storing and processing big data. Used by Internet giants and SMEs alike (the latter often through outsourced services), Hadoop has been transformative in the business sphere with an estimated 76% of Fortune companies implementing the technology by 2015 (Business insider, 2014). Thus we have selected a Hadoop implementation to propose and test a forensic methodology that exemplifies how the above challenges can be addressed through the use of live RAM forensics which facilitate targeted data acquisition.

* Corresponding author. E-mail address: p.leimich@napier.ac.uk (P.Leimich).

We regard the contribution of this paper to be the following:

i. Offer insight into aspects of Hadoop HDFS architecture and how they affect forensic analysis;
ii. Propose a tailored variation of cloud forensic methodology, based on earlier work by various authors, that is based on the findings of contribution i and thus applicable to Hadoop HDFS data breach scenarios (see Section 4.2 and Figure 1);
iii. Provide a case study that evaluates the feasibility of our methodology (contribution ii) with particular focus on the steps that afford initial triage for data acquisition.

2. Background

Complex business and legal demands create impediments to 'in-cloud' forensics that add to the problem of traditional forensic approaches being rendered infeasible by the sheer volume of data, however the question of forensic readiness has not yet been answered. The need for a forensic methodology scalable to the big data age is apparent.

In 'traditional' digital forensic investigations well-established guidelines and methodologies, such as the ACPO guidelines (Association of Chief Police Officers) and DFRWS guidelines (Digital Forensic Research Workshop), are used to safeguard the validity and integrity of evidence and the investigative process as a whole. Traditional methodologies utilise 'dead' acquisition techniques as a means of evidence gathering in which identical bit-to-bit images are produced. However, given that indexing speed decreases as the volume of data increases (Lee and Hong, 2011) this approach is not well suited for big data forensic scenarios such as in a Hadoop cluster. Writing to four external devices simultaneously with a transfer rate of over 6GB/min, it would take 28 days to produce a bit-to-bit image of one petabyte of data (Fowler, 2012). As Fowler (2012) underlines, ideally this image should not be examined directly, but instead used as a master image from which a further copy should be produced for examination, to avoid the risk of irrevocably contaminating the image during an investigation. When factoring in the imaging of the image, the acquisition stage alone of a petabyte of data is 56 days (Fowler, 2012).

When considering forensics, it is important to contextualise Hadoop's adoption within the business sphere, as further challenges become apparent. Although Hadoop can be run "in-house" at a company's own data centre, SMEs will typically not have the facilities or the administrative capacities to maintain their own cluster. In these instances, Hadoop will be used as a Platform as a Service (PaaS) through cloud service providers (CSPs) such as Amazon's EC2 (Grispos et al, 2012). If a data breach were to occur in these instances then the acquisition stage of the forensic investigation also takes on further legal and ethical issues; namely multi-tenancy – multiple clients sharing access to a CSP's data nodes (Barrett and Kipper, 2010; Martini and Choo, 2012), and organisational – the involvement of a third party (the CSP) in the investigation (Ruan et al, 2011). A further legal consideration arises when considering that a CSP's cluster may physically reside in a different country from the breached client, meaning that both parties are governed by different legislation and jurisdictions (Spyridopoulos and Katos, 2011).

Blanket dead acquisition is infeasible when these considerations are made. Indeed, even if an organisation maintains its own datacentre, rendering the legal considerations less applicable, the lengthy process of imaging all of the nodes in the cluster will still cause undesirable downtime resulting in a loss of business (Cho et al, 2012). It is therefore apparent that methodologies such as the ACPO and DFRWS guidelines are unworkable for big data storage environments (Grispos et al, 2012; Hegarty et al, 2012; Lallie and Pimlott, 2012; Martini and Choo, 2012; Poisel et al, 2013), and a new set of both technical and procedural guidelines need to be established to deal with the complex challenges highlighted (Cho et al, 2012; Martini and Choo, 2012).

3. Hadoop Architecture

Hadoop is a Java based system built for UNIX based operating systems. The Hadoop Distributed File System (HDFS) is a master/slave architecture in which DataNodes (slaves) are set up in racks (a grouping of nodes in the same network topology – effectively each rack is a LAN) and clusters (the overarching grouping of racks in an implementation), and receive read/write instructions from the NameNode (master). In HDFS a file is fragmented into blocks which are then distributed among the DataNodes in the cluster. The fragmented file is replicated across different physical addresses in the cluster to provide redundancy in the event of node failure (Apache, 2013). The standard replication factor for data blocks in HDFS is three, meaning that every block will have an original and two further copies placed in the cluster (Apache, 2013). Hadoop is a scalable architecture optimised for a small number of very large files; a single file could be petabytes in size and fragmented into thousands of blocks.

3.1 NameNode

To keep track of the many DataNodes, the NameNode stores a block mapping namespace that contains the physical addresses of each HDFS block within the cluster. It deduces this from information about the blocks in their logical HDFS which is sent periodically by the DataNodes via TCP heartbeat signals (White, 2014). As these block mappings are stored in RAM, they are volatile and the amount of RAM required is proportional to the amount of data stored in HDFS.

The NameNode also stores metadata about HDFS persistently in its local filesystem in two files, FsImage and the Edits log. FsImage is a point in time snapshot of all file system metadata; it contains a serialised form of every data block and HDFS directory in the file system. It also stores metadata pertaining to the Hadoop environment configuration including replication level, block size and modification times (White, 2014). However, FsImage does not contain any physical address mapping information (White, 2014). The Edits log is a transaction log that records changes to the file system metadata in real time, including changes to block-to-file-mappings, block replications and access/modification timestamps. A checkpointing process is used to periodically write these changes back to FsImage (see Section 3.4).

3.2 DataNodes

The DataNodes are the 'workhorses' of HDFS (White, 2014) which store the data blocks allocated by the NameNode via write requests. The data blocks are typically either 64MB or 128MB in size. To HDFS, the cluster file system, each is a replica of a file fragment. However, the DataNode's underlying local file system (e.g. Linux) regards each as a plain file and stores it as such. Therefore, when a data block is deleted by HDFS, its corresponding file will remain in the unallocated space of the DataNode's local file system until overwritten.

3.3 Secondary NameNode / Backup Node

The *Secondary NameNode* can be thought of as the 'checkpointing server' (White, 2014). A designated node within the Hadoop architecture is required to handle checkpointing as the Edits log can grow to be very large in size (potentially 999,999 transactions), making the process both potentially time consuming and memory exhaustive (White, 2014).

From Hadoop 2 onwards, the *Backup Node* replaces the Secondary NameNode. In addition to handling the checkpointing process, it maintains a copy of the file system namespace in RAM, mirroring the NameNode (Apache, 2014).

3.4 Forensic Considerations

The physical HDFS block address namespace is of vital forensic importance, as knowledge of physical block locations in the cluster enables the targeted acquisition of selected DataNodes. As this is stored only in the NameNode's RAM (Spyridopoulos and Katos 2011), it would be lost if dead acquisition techniques were employed on the NameNode. Thus traditional methods in which RAM acquisition is avoided are too comprehensive and fundamental in approach to meet the needs of big data

requirements (Lee et al, 2014). In forensic investigations carried out on a Hadoop 2.4.1 instance, it would be advantageous to perform RAM acquisition of the Backup Node instead of the NameNode. This would minimise interaction with the NameNode whilst in a live state, whilst also providing the potential to acquire a perfect replica of the namespace mapping.

Forensically the FsImage file, Edits log, and checkpointing process are also important components of HDFS. Information of forensic value that can be deduced from FsImage includes the block-to-file mappings, replication level, and modification/access times. However, if much time has elapsed since the last checkpoint, FsImage could be vastly outdated and the Edits log could potentially contain hundreds of thousands of transactions that have not been recorded in real time by FsImage. Default HDFS settings trigger an automatic checkpoint after 1,000,000 transactions or one hour, whichever comes first (Apache, 2013, Wang, 2014). If fifty-five minutes had elapsed since the last checkpoint and the Hadoop cluster had an average of 500 transactions per minute, the FsImage file would be out of date by 27,000 transactions which are recorded in the Edits log only. To ensure that the most recent state of the file system is used during an acquisition, Fowler (2012) advocates the use of checkpointing prior to acquiring physical data from the nodes. A copy of the original Edits log should be retained to enable the investigator to examine the most recent changes directly.

4. Existing Research and Practices in DFS and Cloud Forensic Acquisition

This section summarises selected works by a number of authors who have discussed forensic methods for data breaches in distributed file system or cloud contexts, and discusses their applicability to the specific context of Hadoop HDFS.

Spyridopoulos and Katos (2011) present an analysis of the Google File System (GFS) and outline a template for a tailored forensic recovery tool. They point out that, in DFS data breach scenarios, evidence manifests as either content or non-content based. The former includes slaves storing the target data while transaction logs and metadata fall into the latter category and are usually found on the master. A suitable methodology needs to access both forms of evidence; Spyridopoulos and Katos (2011) address this by delving into the metadata and log files of the master node, as well as unallocated space in the data nodes. Spyridopoulos and Katos (2011) classify DFS data breach scenarios into four categories and discuss the forensic challenges posed by each. The first category, where the attacked data is live, i.e. has not been deleted, and the cluster is held in-house or within the investigation's jurisdiction, does not pose any particular problems; standard forensic procedures can be applied. They point out that the second category, where data is live but resides outside the jurisdiction, can be reduced to a first category problem by leveraging the file systems replication mechanisms. Via the client's computer, working slaves can be set up within the jurisdiction. By deliberately causing the slaves outside the jurisdiction to fail, the file system is induced to replicate the data blocks onto the working slaves, thus transferring them into the jurisdiction. Spyridopoulos and Katos (2011) distinguish the different perspectives of the master and slaves: a block is a fragment of a user file to HDFS and regarded as such by the master, but the slaves' local file system regards the block as a file. This is important in cases where the data in question has been permanently deleted, because it means that the block will persist locally in unallocated space until it is overwritten. Therefore, standard forensic data carving methods can be used to recover deleted blocks provided it is known which slave the data can be found on and this slave is within the jurisdiction (third category). Additionally, the physical offset on the slave's disk needs to be known in order to avoid the privacy issues that would result from having to acquire an entire slave in multi-tenancy situations. In the fourth category, where data is deleted and resides outside the jurisdiction, any investigation relies on international legal agreements. Spyridopoulos and Katos (2011) summarise their high-level method in a flow chart. After identification of evidence sources (master's RAM, log files, slaves), this is analysed to check whether data has been deleted, followed by establishing whether the blocks are inside jurisdiction. If yes, the blocks can be acquired from the local filesystem and reconstructed. However, as Spyridopoulos and Katos (2011) point out, in the case of deleted data, acquisition additionally relies on

knowledge of the block to last chunk metadata, which may be available from the slave's local file system until it is overwritten. In case of the latter, the data is lost.

The example presented by Spyridopoulos and Katos (2011) shows an acquisition walkthrough for an in-house scenario with two small files of interest, one of which has been deleted, and thus combines their first and third category. The setup comprises four slaves which physically reside on the same machine but use different ports for communication with the master, and uses the Cloudstore distributed file system which is similar to GFS. They present a detailed analysis of the master node's log files to determine the data nodes on which the blocks in question are/were stored, and proceed to reconstruct the files, but provide little detail as to how they obtain the block to last chunk information.

Spyridopoulos and Katos conclude with a gap analysis. For cloud storage systems to be forensically ready, four issues need to be addressed: 1. Legal agreements between involved countries, 2. Legislation for digital data retention upon deletion, 3. The block to last chunk mapping must be implemented in the cloud architecture and 4. Implementation of persistent storage of chunk locations in file records. While all of these suggestions are desirable, they are outwith the control of forensic investigators or clients.

Cho et al (2012) present a high level method specifically for Hadoop based systems, which follows the standard process of preparation, identification, collection, analysis and reporting. Their method iterates through collection and analysis twice, firstly capturing the volatile data held in the NameNode using a combination of live and dead acquisition methods and performing real-time analysis of the NameNode's FsImage and Edits log to ascertain which of the DataNodes store the data in question. This is followed by imaging the DataNodes with dead acquisition techniques and analysing the resulting artefacts. Cho et al (2012) do not present a case study or specific scenario to illustrate their proposals. It could be argued that the use of dead acquisition on entire data nodes does not adhere to the pragmatic considerations and ethical dimensions outlined in Section 2.

Fowler (2012) does not present a methodology as such, but discusses HDFS architecture through the eyes of a forensic analyst, pinpointing the most important artefacts for an investigation including RAM, FsImage and Edits log, and emphasising the forensic importance of manually triggering a checkpoint (see Section 3.4). Fowler (2012) also discusses the potential value of the HDFS Trash, which is disabled by default (Apache, 2013, 2014), but often enabled by administrators as it is recommended for example by Cloudera (2015). If enabled, it means that files are not fully removed upon deletion, but moved to trash where they remain available for a set period. However, even when trash is enabled, deleted blocks may be not found there because (a) the time limit may have been exceeded, (b) using the '*hadoop fs -rm -r'* command with the *'-skipTrash'* option provides an easy anti-forensic measure and (c) programmatic deletion via third party interfaces would use the trash only if this functionality is specifically provided by the interface (Cloudera, 2015).

Martini and Choo (2012) present an integrated iterative conceptual cloud forensic framework that focuses on the preservation and collection of cloud computing data for forensic purposes and aims to address the issues discussed in Section 2. The framework comprises four phases, namely 1. Evidence source identification and preservation, 2. Collection, 3. Examination and analysis and 4. Reporting and presentation. This appears similar to traditional digital forensic frameworks, but differs in iterating through phases 1-3 several times, using the information from each iteration in the next. Martini and Choo (2012) also emphasise that evidence preservation should commence as soon as the use of cloud services has been identified in order to account for the cloud's ever changing environment. Martini and Choo (2014a; 2014b) carry out technical experiments to validate the method, using XtreemFS and VMware vCloud respectively as case studies. The number of iterations and their specific purpose vary; for XtreemFS, which is more closely related to HDFS than VMware vCloud, three iterations are used relating to evidence from the directory service, the metadata and replica catalog and the object storage device respectively. The latter correlates with the DataNodes in HDFS while the former two correspond to various aspects of information stored in the NameNode's log files and RAM.

The design of our methodology (see Section 5.2) is based on the work presented above, namely Martini and Choo (2012; 2014a; 2014b), Spyridopoulos and Katos (2011), Fowler (2012), Cho et al (2012) and Patrascu and Patriciu (2014).

5. Design of Methodology

5.1 Requirements

In order to reflect Hadoop's functionality, business implementation context, and the time constraints of the forensic acquisition process, a suitable forensic methodology must have the following characteristics:

Scalable - Given that Hadoop implementations can range from a single standalone machine to clusters containing hundreds or more machines it is of fundamental importance that the methodology be scalable to meet the requirements of data centres of varying sizes. A methodology must also be scalable to the amount of data that is held within the cluster, as Hadoop implementations commonly host petabytes of data.

Business Aware – As Hadoop is typically implemented within a business context, a methodology needs to be conscious of the fact that the breached party has already been inconvenienced by the attack and should minimise further business disruption. The use of targeted acquisition is one such way to achieve this by reducing node down time at the start of the investigation.

Targeted – To reduce acquisition time and minimise business disruption to the breached party, it is desirable to image only the DataNodes storing the data replicas relevant to the investigation. This move away from traditional blanket acquisition requires foreknowledge of what data has been affected by the data breach and where this data physically resides in the cluster.

Use Live/Dead Acquisition Techniques – Only live acquisition of the NameNode can capture the RAM which stores the physical HDFS block address namespace necessary for targeted acquisition of the DataNodes (see Section 3.4). There is no need for live acquisition of DataNodes as the data is stored persistently and the system would use replicas of the data to maintain availability while DataNodes are taken off-line one by one.

Reliable – To be reliable, a forensic investigation must use up to date information. Checkpointing is suggested as a means of ensuring that the metadata acquired from FsImage is an accurate and up-to-date representation of the file system at the time of acquisition (see Section 3.4).

Minimise Cluster Interaction – Direct interaction with all nodes should be kept to a minimum to reduce the risk of contamination not just of the evidence, but also of the target company's data in the cluster. Additionally, a data centre may be located at considerable distance from the affected client and/or access may be controlled by the CSP, restricting the time available to the investigator in which they can physically enter the data centre to perform their acquisition tasks.

Applicable to Local/Outsourced Implementation – Given that Hadoop implementations can exist both as local data centres and outsourced via CSPs, a methodology needs to exist which is implementable in both scenarios.

5.2 Design

The proposed forensic methodology for Hadoop data breach scenarios is based on existing work summarised in Section 4 and meets the requirements discussed in Section 5.1. It comprises nine phases which are illustrated in Figure 1 and detailed in the following subsections.

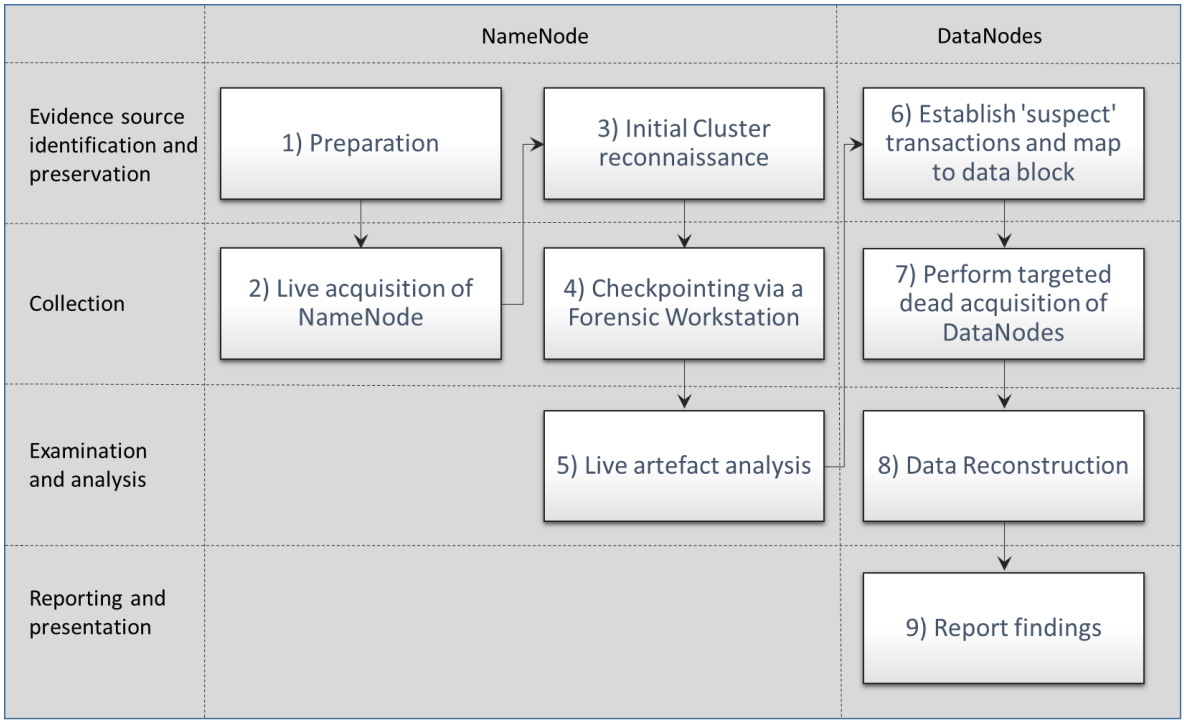

Figure 1. Proposed forensic methodology. The nine phases are described in Section 5.2 and applied to a test scenario in Section 6.2. Labels on the left match those used by Martini and Choo (2012, 2014a) for comparison, while labels at the top identify the corresponding nodes.

Phase 1 – Preparation

To prepare for the forensic investigation, the examiner needs to identify basic properties of the HDFS to be investigated such as the NameNode's address and its jurisdiction, as well as gaining access by obtaining suitable credentials.

Phase 2 – Live acquisition of NameNode RAM and volatile artefacts

The NameNode's RAM namespace facilitates the identification of the physical offset mappings of HDFS data blocks on the DataNodes. Live acquisition of the NameNode is of vital importance to avoid losing this information, which is crucial as the methodology must be able to distinguish areas affected by the attack in order to facilitate targeted acquisition (Sections 3.4 and 4). Time is of the essence for RAM capture as heartbeat signals are sent only by live data blocks, meaning that our chances of retrieving the vital offsets sent via these heartbeats will decline rapidly after data deletion. Placing this Phase before the cluster reconnaissance is in line with Martini and Choo (2014b).

At this stage the FsImage and Edits log files should also be preserved and copied from the NameNode to a forensic workstation. Although these files are stored persistently, the Edits log is highly volatile due to the fact that it records all cluster transactions in real time. Whilst less volatile than the Edits log, the FsImage file, which contains the HDFS tree and metadata, is still a volatile artefact as it is expected to be modified at least hourly by the automatic checkpointing process (Section 3.4).

Phase 3 – Initial cluster reconnaissance

The purpose of this phase is to acquire knowledge of file system metadata that provides useful information for how the investigation should progress, such as block size, replication factor and DataNode associated with given block IDs. This phase is independent of the NameNode RAM acquisition and could be carried out first, however, even the shortest delay to RAM acquisition could risk losing vital information. Any commands issued could affect the RAM, risking forensically undesirable contamination of potential evidence.

This phase uses Hadoop inbuilt commands, primarily designed for cluster administration, which are available in the live system through the Hadoop web user interface and various inbuilt command line scripts (Fowler, 2012). Of the numerous command scripts that could be used in this phase we recommend three in particular: the Hadoop file system checking utility '*hadoop fsck'*, the '*dfsadmin -report*' script and the OIV image viewer. Detailed information about these commands can be found in Section 6.2 (Phase 3) and sample output in Figures 4a, 4b and 5 respectively.

Fowler (2012) recommends using inbuilt command-line scripts such as these as they give complete information while the web user interface shows only a selected subset. These commands should be grouped in pre-programmed and exhaustively tested prepared scripts and executed remotely from a forensic workstation in order to meet the requirement of minimising interaction with the cluster, particularly the NameNode, and mitigating the risk of evidence contamination by ensuring that any effect on the cluster is entirely predictable.

Phase 4 – Checkpointing via a forensic workstation

Fowler (2012) recognised that unless the examiner performs a checkpoint operation prior to acquiring and analysing the FsImage and Edits log, the metadata contained within the FsImage has the potential to be vastly out of sync with the accurate state of the file system (see Section 3.4). However, doing so risks contaminating the NameNode's volatile memory as java processes to transfer the FsImage file to the Secondary NameNode are invoked.

To alleviate the risk of data contamination within the live cluster, and in line with the principle of minimising cluster interaction during the investigation, we carry out the checkpointing process outside the cluster on a forensic workstation with Hadoop configured in pseudo-distributed mode. Copies of the FsImage and Edits logs collected in Phase 2 are placed on the forensic workstation and the checkpoint triggered using the inbuilt command *'checkpoint -force'*. The checkpoint operation flushes the Edits log out to the FsImage file in order to update the state of the HDFS namespace. All of the previously unmapped transactions are mapped into the FsImage file, updating its information on the HDFS namespace, and the Edits log is emptied. However, as the transactions themselves are of high forensic value in an investigation, it is recommended that a copy of both original files is made on the forensic workstation before triggering the checkpoint process. This ensures that the examiner has at her disposal both the updated view of the HDFS namespace and a record of all transactions that occurred between the time of the previous checkpoint operation and the attack (see Phase 6).

Patrascu and Patriciu (2014) suggest the use of a hypervisor layer to aid in the processing of forensic artefacts by running MapReduce scripts on virtual machines. MapReduce is a parallel processing framework which achieves speed gains of several orders of magnitude in processing data stored in HDFS by distributing tasks across a number of slaves (White, 2014). The hypervisor approach is an innovative use of virtualisation, and such a process could well be followed as an alternative method to aid checkpointing outside of the cluster.

Phase 5 – Live artefact analysis

During this phase the artefacts gathered in Phases 2 and 4 are analysed in order to establish first which data blocks have been affected by the attack (in our case deleted). Up until this stage the investigator (and indeed breached party) may have no inclination as to what the specific blocks that were targeted in the attack. By performing analysis on the NameNode RAM image acquired in Phase 2, the examiner is able to identify the blockIDs and startOffset values targeted in the attack. These values are significant with HDFS because they relate to the internal blockID given to the specified HDFS data block, and the physical start offset address that the located at within the DataNode's storage. These values are of key importance to our methodology as they enable later targeted acquisition of the DataNodes.

Further to the analysis of the NameNode RAM, the FsImage file should also be analysed in this phase. Following the checkpoint conducted in Phase 4, the file should contain an up to date record of the HDFS namespace. This is of forensic value as it will provide the examiner with an idea of the basic tree structure of the target HDFS directory, and give context to the investigation at end. Further to this, differential analysis between the pre and post checkpoint versions of the FsImage file (as discussed in Phase 4) can provide a beneficial first point of call to identify any obvious discrepancies that occurred to the HDFS tree following the time of the attack.

Phase 6 – Establish suspect transactions and map to data blocks

This phase can reveal evidence of attack provenance through cross-referencing with scheduled delete and copy requests from HDFS (Fowler, 2012). Furthermore, this stage may aid scoping the attack by uncovering HDFS data blocks affected in the attack that had not been previously identified as such.

The copy of the original Edits log is analysed to establish suspect transactions which took place during the attack timeframe. Creating a copy of the Edits log and FsImage file prior to the checkpoint operation on the forensic workstation enables complete analysis of all transactions even after the Edits log has been flushed during the checkpoint (see Phase 4).

As well as analysing the Edits log, the examiner should also continue their analysis of the NameNode RAM capture for corroborating evidence. In the previous phase the examiner was able to establish through the RAM capture what blocks had been targeted in the attack, this phase expands the analysis by seeking to establish what happened to those blocks during the attack.

Phase 7 – Perform targeted dead acquisition of DataNodes

By this phase of the investigation it is anticipated that the examiner will have established which blocks were affected by the attack and where they physically reside in the cluster, meaning that acquisition can now commence on the DataNodes themselves. Knowledge of the findings of previous stages enables the investigator to establish a minimal set of DataNodes which comprises at least one replica of each file or block suspected to be affected by the attack. This enables the investigator to selectively target only the required DataNodes for imaging. Shutting down one DataNode at a time minimises business disruption as the cluster as a whole will remain operational thanks to the inbuilt redundancy of HDFS data storage.

If the start offset of a target block for acquisition is known in combination with the block size, there is potential for an even more targeted approach, where only the relevant blocks are imaged rather than the whole DataNode. This is akin to on-the-fly file carving. However, it should be noted that whilst this method would be ideal, it may not always be possible due to local file system data fragmentation.

Phase 8 – Data reconstruction

Once the DataNodes have been imaged they can be analysed for traces of deleted data blocks which can then be reconstructed through data carving. Despite reducing the volume of the data to be analysed through selective imaging of the nodes, this is likely to be a resource and time exhaustive procedure. One option for increasing the efficiency of this phase could be to load the raw disk images into a forensic HDFS installation and use MapReduce to aid the processing of the data (Carrier, 2015; Hegarty et al, 2012; Lee and Hong, 2011; Lee et al, 2014; Lee and Un, 2012; Patrascu and Patriciu, 2014).

Phase 9 – Report findings

As with all forensic investigations this methodology culminates in reporting the findings to the relevant stake holders.

6. Testing

6.1 Scenario Preparation

In order to test the feasibility of the proposed methodology, a small Hadoop cluster comprising one master and three slaves was configured in fully distributed mode on commodity hardware running Ubunutu 14.04 (see Figure 2). Each machine had 4GB RAM and Intel Dual Core processors.

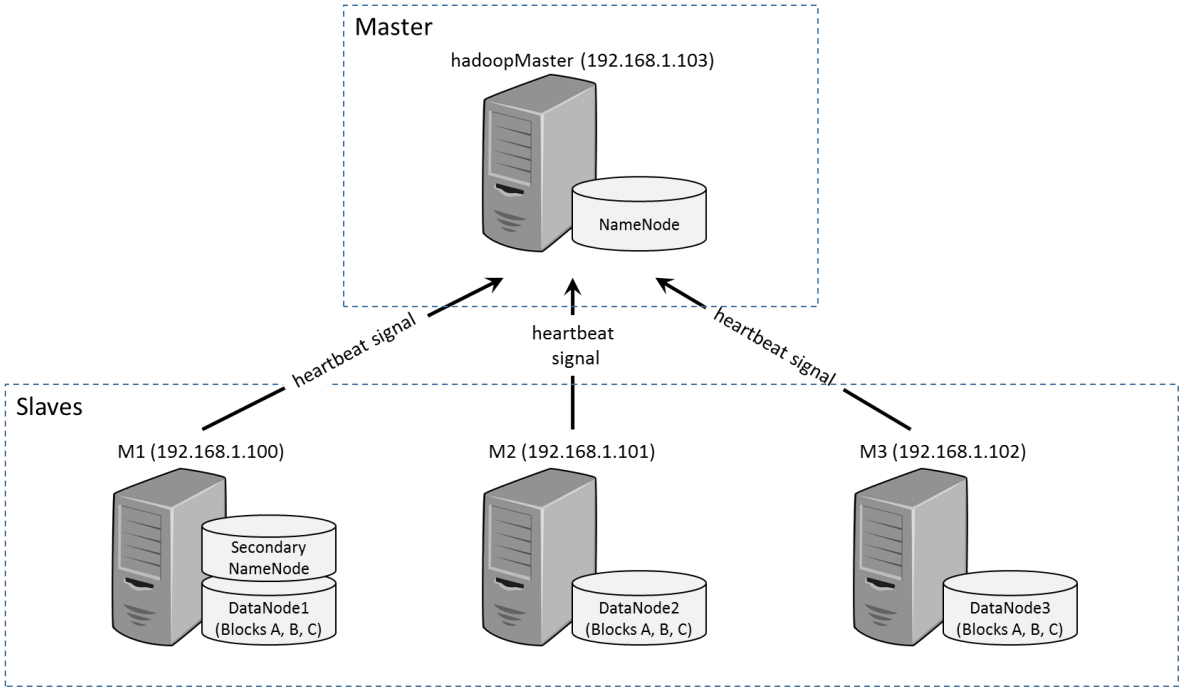

Figure 2. Network topology of the Hadoop cluster used for testing, including IP addresses, hostnames, nodes and replicated data blocks A, B, C.

Hadoop 1.2.1 was used for the purpose of this experiment. Although there are newer 2.x.x additions, 1.x.x is not redundant; many organisations choose to implement this version as it best suits their requirements. Indeed, all of Amazon's EMR Hadoop platforms are offered both as 2.x.x and 1.x.x implementations. Given the relative infancy of the field of Hadoop forensics, it was felt prudent to focus attention on the original version of the platform in this instance.

Spyridopoulos and Katos (2011) categorise HDFS data breach attacks in two dimensions. The first dimension distinguishes by jurisdiction, between attacks where the breached data resides within the physical and legal jurisdiction of the breached party (i.e. an attack on a cluster managed internally by the breached party) and attacks in which the breached data is outside of the jurisdiction of the breached party (for example in hosted cloud services such as AWS). By using an in-house cluster, our test scenario fits into the former category. It is designed as a proof of concept, allowing us to establish the feasibility of our methodology in principle before considering its scalability and applicability to the more problematic area of outsourced services. The second dimension considered by Spyridopoulos and Katos (2011) concerns the state of the breached data after the attack, which could be live (in the case of unauthorised access or modification) or deleted. As discussed in Section 4, the latter category poses additional challenges particularly when the block metadata may also have been overwritten. We therefore devised a scenario in line with the latter, simulating a situation whereby data had been illicitly deleted from the Hadoop cluster. If selective acquisition of blocks from NameNodes is shown to be feasible in this 'worst case', the further work required to extend this study to include live data should be minimal.

To obtain sample data to store on the cluster, we downloaded the 163.6MB file 'enwiki-latest-pages-articles1.xml' from Wikipedia. This is one of the publically available datasets available at *'https://dumps.wikipedia.ord/enwiki/latest'*. The inbuilt Hadoop command *'hadoop fs –moveFromLocal [source | destination]'* was used to push the file from the local file system to be distributed into HDFS (Apache, 2013). The default block size (64MB) and block replication factor (three) were used. Having issued the inbuilt command '*dfsadmin –report'* to confirm that the file was successfully distributed and replicated across HDFS and the cluster was in a healthy state, the deletion scenario was implemented. The data was deleted from HDFS with the command '*hadoop fs -rm [target file]'* (Apache, 2013). The success of the deletion was verified via the Hadoop web UI (see Figure 3).

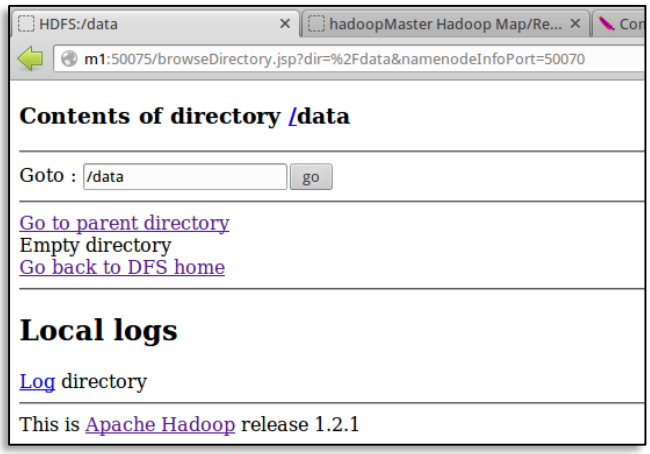

Figure 3. Hadoop web UI output post deletion (simulated attack) showing the empty directory from the NameNode.

6.2 Forensic Process - Application of Methodology

Having created forensic artefacts for investigation in the cluster, the forensic process could commence. The designed methodology (see Section 5.2) was used as a guideline for the procedure undertaken, and this section shall discuss the methods used in each phase. The focus was placed on NameNode RAM acquisition and analysis in order to ascertain whether targeted acquisition of DataNodes would be possible based on the results found; a full forensic investigation of the DataNode images was beyond the scope of this investigation.

Phase 1 – Preparation

This phase is not required in this proof of concept; as creators of the scenario we already had the required knowledge and access.

Phase 2 – Live Acquisition of NameNode

The Linux memory acquisition tool *fmem* was used image the NameNode's RAM. This tool was chosen as it circumvents the kernel protection which otherwise prevents access to the /dev/mem directory that contains the system RAM. This restriction applies even to users with root permissions (Ubuntu, 2015). Once fmem has been installed and compiled, it can be invoked in with the syntax *'sudo dd if=/dev/fmem of=[output directory] bs=512'*. In our instance this created a 3.2GB image of the NameNode's RAM.

Phase 3 – Initial Cluster Reconnaissance

Initial cluster reconnaissance was performed using the three inbuilt Hadoop command scripts introduced in Section 5.2. As explained there, this is carried out after the RAM acquisition in order to avoid contamination of the RAM. The Hadoop filesystem checking utility *'hadoop fsck -blocks*' (Apache, 2013) was invoked first to obtain high-level information about the cluster; the *'-blocks'* option provides a block report. The output is shown in Figure 4a. The utility has three further optional parameters which can be used to tailor the output: *'-rack'* to provide a network topology of the cluster, *'-files'* to

list the files within the cluster and *'-locations'.* The latter shows which DataNodes each data block is stored on. However, this only provides a high level block-to-node mapping which does not include the physical start offset locations of the data HDFS blocks; it also proved of no value to our investigation as it includes only current, not deleted blocks. Secondly, additional information of forensic value was obtained with the '*dfsadmin -report*' command script (Apache, 2013) which returns a detailed snapshot of the state of the HDFS cluster that includes DataNode IP addresses, configured node/ cluster capacity and percentage of disk space used/available. Example output is shown in Figure 4b.

Thirdly, the OIV (offline image viewer) is an inbuilt HDFS administrative utility which enables a human readable view of the HDFS tree from the raw hex FsImage file (Apache, 2013). Figure 5 shows the output from the OIV command issued with the syntax '*oiv -i [input file] -o [output directory]*'. Comparing the OIV output with the DFS in use (40KB per DataNode) shown in Figure 4a, we can conclude that the highlighted file (approx 163MB) must have been deleted from HDFS.

(a)

```
student@hadoopMaster:~$ hadoop fsck -blocks
FSCK started by student from /192.168.1.103 for
path / at Tue Mar 03 16:47:22 GMT 2015
.Status: HEALTHY
 Total size:  4 B
 Total dirs:  8
 Total files: 1
 Total blocks (validated):    1 (avg. block size 4 B)
 Minimally replicated blocks: 1 (100.0%)
 Over-replicated blocks:      0 (0.0%)
 Under-replicated blocks:     0 (0.0%)
 Mis-replicated blocks:       0 (0.0%)
 Default replication factor:  3
 Average block replication:   3.0
 Corrupt blocks:              0
 Missing replicas:            0 (0.0%)
 Number of data-nodes:        3
 Number of racks:             1
FSCK ended at Tue Mar 03 16:47:22 GMT 2015 in 11
milliseconds

The filesystem under path '/' is HEALTHY
```

(b)

```
student@hadoopMaster:~$ hadoop dfsadmin –report
Configured Capacity: 286855438336 (267.15 GB)
Present Capacity: 262510673920 (244.48 GB)
DFS Remaining: 262510551040 (244.48 GB)
DFS Used: 122880 (120 KB)
DFS Used%: 0%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0

---------------------
Datanodes available: 3 (3 total, 0 dead)

Name: 192.168.1.100:50010
Decommission Status: Normal
Configured Capacity: 75515535360 (70.33 GB)
DFS Used: 40960 (40 KB)
Non DFS Used: 7199408128 (6.7 GB)
DFS Remaining: 68316086272 (63.62 GB)
DFS Used%: 0%
DFS Remaining%: 90.47%
Last contact: Tue Mar 03 16:48:25 GMT 2015

Name: 192.168.1.101:50010
Decommission Status: Normal
Configured Capacity: 60247990272 (56.11 GB)
DFS Used: 40960 (40 KB)
Non DFS Used: 6239903744 (5.81 GB)
DFS Remaining: 54008045568 (50.3 GB)
DFS Used%: 0%
DFS Remaining%: 89.64%
Last contact: Tue Mar 03 16:48:25 GMT 2015

Name: 192.168.1.102:50010
Decommission Status: Normal
Configured Capacity: 151091912704 (140.72 GB)
DFS Used: 40960 (40 KB)
Non DFS Used: 10905452544 (10.16 GB)
DFS Remaining: 140186419200 (130.56 GB)
DFS Used%: 0%
DFS Remaining%: 92.78%
Last contact: Tue Mar 03 16:48:25 GMT 2015
```

Figure 4(a). Output of the command 'hadoop fsck –blocks' showing default replication factor, number of corrupt blocks, number of racks in the cluster and other information of potential forensic value.
(b). Output from the 'dfsadmin –report' command issued post deletion of HDFS data. The first paragraph shows the system summary including total configured cluster size; values given are the sum of the corresponding DataNode values. The line 'Decommission Status: Normal' denotes that a node is live. Other forensically valuable information includes the DataNode IP addresses, HDFS storage capacity and percentage of HDFS used.

```
drwxr-xr-x - student supergroup          0 2015-02-25 18:25 /
drwxr-xr-x - student supergroup          0 2015-02-25 18:25 /data
drwxr-xr-x - student supergroup          0 2015-02-17 09:44 /usr
-rw-r--r-- 3 student supergroup 163613798 2015-02-25 18:25 /data/enwiki-latest-pages-articles1.xml-p000000010p000010000
drwxr-xr-x - student supergroup          0 2015-02-17 09:44 /usr/local
drwxr-xr-x - student supergroup          0 2015-02-18 10:22 /usr/local/hadoop
drwxr-xr-x - student supergroup          0 2015-02-25 18:21 /usr/local/hadoop/tmp
drwxr-xr-x - student supergroup          0 2015-02-26 15:11 /usr/local/hadoop/tmp/mapred
drwx------ - student supergroup          0 2015-02-26 15:11 /usr/local/hadoop/tmp/mapred/system
-rw------- 3 student supergroup          0 2015-02-26 15:11 /usr/local/hadoop/tmp/mapred/system/jobtracker.info
```

Figure 5. Output from running the OIV command against the FsImage file stored on the NameNode. Highlighted is the Wikipedia XML file that was stored in HDFS. Reading from left to right: file permissions, replication factor (3), owner, group, total file size, file access date and time, and finally the file path in HDFS.

Phase 4 - Checkpointing via a Forensic Workstation

Due to the fact that the cluster was in use only for this experiment, and it was categorically known that no further transactions had taken place, this phase could be omitted. However, as discussed in Sections 3.4 and 5.2 (Phase 4), it is strongly recommended that this phase be conducted in non-experimental environments.

Phase 5 – Live artefact analysis

The aim of this phase is to establish the HDFS namespace containing the block-to-offset mapping to facilitate targeted DataNode acquisition. Initially the captured RAM image was manually viewed in the Linux hex editor *Bless* to search for evidence strings from the deleted Wikipedia file. As a result of this manual analysis of the RAM capture, evidence of data block IDs was found. Figure 6 shows two examples.

M1 block without startOffset:

```
http://m1:50075/browseBlock.jsp?blockId=2724904701271491951&
blockSize=67108864&filename=%2Fdata%2Fenwiki-latest-pages-ar
ticles1.xml-p000000010p000010000&datanodePort=50010&genstamp
=1004&namenodeInfoPort=50070&chunkSizeToView=32768
```

M3 block with startOffset:

```
http://m3:50075/browseBlock.jsp?blockId=2724904701271491951&
blockSize=67108864&startOffset=98304&genstamp=1004&filename=
%2Fdata%2Fenwiki-latest-pages-articles1.xml-000000010p000010
000&chunkSizeToView=32768&datanodePort=50010&namenodeInfoPor
t=50070
```

Figure 6. Two extracts from the RAM image of the NameNode showing examples of data block information stored. Differences are highlighted in bold and underlined. Both examples show information about the same block, replicated on nodes M1 and M3 respectively (note the identical block ID). The URI identifying each DataNode and its default port is followed by the universal string 'browseBlock.jsp?blockId='. The headers shown contain the same information, except for ordering, and one crucial difference: the startOffset address is included for the block replica on M3, but missing for the replica on M1.

Figure 6 also illustrates that the string *'browseBlock.jsp?blockId='* was found to directly precede each data block ID stored in RAM. Viewing the Hadoop source code revealed that browseBlock.jsp is a process that informs the NameNode's block mapping namespace. This confirms that the string *'browseBlock.jsp?blockId='* or its hex equivalent will reliably precede every block ID and can therefore be used as a *magic number* to search for references to HDFS blocks during NameNode RAM analysis. A Python script was developed to automate the search for instances of this string from the image and extract the corresponding block information. This process helped to reduce a great deal of noise from the original RAM image, and could aid the speed of future searches. The results file could be processed further to filter the output by DataNode.

Phase 6 – Establish 'suspect' transactions and map to data block

The Python script described in Phase 5 above was modified in to search for the strings "deletion" and "deleted", and run against the DataNode subsets of the original RAM image gathered in the previous phase. The string 'deletion' was found; this made it possible to establish scheduled data block deletions. The results are presented and discussed in Section 7.2 and Figure 8.

As the focus of our case study was on the RAM analysis, the suggested complementary analysis of the original Edits log (see Section 5.2) was omitted. However, it is expected that this would provide valuable corroboration and additional evidence particularly in an investigation against a large cluster.

Phase 7 – Perform Targeted Dead Acquisition of DataNodes

As described earlier, knowledge of start offset information for data blocks can facilitate targeted acquisition on the DataNodes. In our test, start offset information was only found for blocks stored on M3 (see Figure 6), and therefore this was the machine chosen to be imaged. The standard Linux memory imaging tool *dd* was used to create a physical bit-to-bit copy of M3's disk space.

Phase 8 – Data reconstruction

This phase was outside of the scope of this project, which chose to focus on the NameNode RAM element of the methodology.

Phase 9 – Report findings

The results are listed and discussed in the following section.

7. Results

7.1 NameNode RAM Analysis

As discussed in Section 6.2 (Phase 5) and shown in Figure 6, magic numbers were found for evidence of block records in the NameNode's RAM. The remaining headers in each block record were deduced and are summarised in Figure 7.

URI	BlockID	Block Size	StartOffset	GenStamp	Filename	Chunksize to view	DataNode Port	NameNode Info port

Figure 7. Headers from the browseBlock.jsp process trace found in RAM. Two example process traces are shown in Figure 6.

Having established that browseBlock.jsp contained a record to the block ID and start offset values for HDFS blocks, the Python script described in Section 6.2 was used to carve the RAM image for instances of this string and map it to the known blocks affected by the deletion. Table 1 illustrates the Block IDs (Blocks A, B, C) and startOffset values that were detected in the NameNode's RAM image in relation to each DataNode (M1, M2, M3) in the cluster.

Node	M1		M2		M3	
Block	BlockId found?	StartOffset found?	BlockId found?	StartOffset found?	BlockId found?	StartOffset found?
A	Yes	No	No	No	Yes	Yes
B	No	No	Yes	No	Yes	Yes
C	No	No	No	No	No	No

Table 1. NameNode RAM analysis summary depicting which blocks were found associated with which DataNodes.

7.2 Evidence of Deletions

Given that Phase 6 of the proposed methodology was to 'Establish suspect transactions and map to data blocks', evidence of block deletion was searched for using the Python script discussed in Section 6.2. The RAM extract shown in Figure 8 demonstrates that records of scheduled deletion have a

common format which includes the time and date of the scheduled deletion, the string *'INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling blk_'* followed by the blockID and a little later by the string *'for deletion'*. As these two strings occur in every trace, they, or their hex equivalents, can be used as magic numbers to automate the search for evidence of deletions. It is interesting to note that all three blocks affected by deletion were found within this section of the RAM extract, despite no evidence of Block C (Block ID 539013…) being found in any of the extracted browseBlock.jsp evidence.

```
doop.hdfs.server.datanode.Data
Node: Scheduling blk_272490470
1271491951_1004 file /usr/loca
l/hadoop/tmp/dfs/dat..current/
blk_2724904701271491951 for de
letion. on Line: 3465725 ..Rea
d Line: 2015-03-03 16:48:53,42
9 INFO org.apache.hadoop.hdfs.
server.datanode.DataNode: Sche
duling blk_7567747605822618193
_1004 file /usr/local/hadoop/t
mp/dfs/data/current/blk_756774
7605822618193 for deletion. On
 Line: 3465727 ..Read Line: 20
15-03-03 16:48:53,429 INFO org
.apache.hadoop.hdfs.server.dat
anode.DataNode: Scheduling blk
_5390134442919737633_1004 file
 /usr/local/hadoop/tmp/dfs/dat
..current/blk_5390134442919737
633 for deletion. on Line: 346
```

Figure 8. Extract from the NameNode's RAM image highlighting evidence of all three HDFS data blocks being scheduled for deletion. The strings 'INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling blk_' and 'for deletion' are universal, occurring in every trace.

7.3 Complete and Incomplete browseBlock.jsp Headers

As discussed in Sections 6.2 and 7.2, 'browseBlock.jsp' was identified as a signature string which could be used as a magic number to aid forensic searching of block references in the NameNode's RAM. Using this method, evidence of 44% of block IDs was found in the RAM image (see Table 1). However only half of these records were complete and included the startOffset header and value. The other half lacked this and thus were incomplete. Figure 6 exemplifies this, it shows an incomplete example at the top and a complete example below. Complete and incomplete records differed only in the presence or absence of the startOffset; all other headers (listed in Figure 7) were present in all records.

Given that the block replication factor was set to three and there were three blocks affected by the deletion, there were nine block IDs and nine separate startOffset values to carve for in the NameNode's RAM. Of these 44% (4/9) of block IDs were found referenced in the NameNode's RAM, and only 22% (2/9) of the startOffset values were found. Out of a combined total of 18 possible block IDs and startOffsets, only six were found referenced in the NameNode's RAM image. To put this another way, two thirds were undetected, meaning that whilst some evidence of forensic importance was extracted by this analysis, the majority of HDFS metadata anticipated to be found were not uncovered.

8. Discussion and Evaluation

8.1 Discussion of Findings

The fact that only 44% of block IDs were found in the RAM image can potentially be explained by the DataNodes' heartbeat process. Once a block has been deleted from HDFS, information about it can no longer be sent in the heartbeats. Given the volatile nature of RAM storage, when the NameNode receives a heartbeat signal from a DataNode the information pertaining to HDFS data blocks will dynamically overwrite what it already contains in its RAM store, meaning that the namespace contained is constantly evolving. If we apply this logic to the results found, or more specifically the data blocks *not* found in the RAM image, we can begin to form a theory based on individual component difference within the nodes, which can offer an explanation to the sparseness of results. Further research is required to test this theory by analysing not just the contents of a NameNode's RAM, but also investigating the provenance of this information by examining the DataNode's internal Java processes that led to the propagation of information sent in the heartbeat packets. This would aid in better understanding how it is that the NameNode's RAM namespace is generated and maintained.

The results also unearthed the string 'INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling blk_' which was present in the NameNode's RAM in relation to every block that was affected by a deletion transaction. This string, or its hex equivalent, could provide a very useful magic number, as it is clearly specific to a Hadoop deletion and therefore could avail useful forensic information. However, it should be noted that in real world scenarios using this string alone may also avail false positives as all deletions, whether genuine or illicit, will be output. If the block ID of the block illicitly deleted is known, then this could be used in combination with this string to filter for only relevant results.

Although the results of 44% of BlockIDs and 22% of startOffset values found present in the NameNode's RAM image appear too inconclusive to validate the success of the methodology fully, pragmatically speaking, in a 'real world' scenario these results would provide a means for further targeted acquisition on the DataNodes. The examiner would be able to target specifically the nodes in the cluster which had been identified as containing HDFS blocks scheduled for deletion. As the cluster size is scaled up, the investigative time reduction for both acquisition and analysis is exponential when the investigator is able to target DataNode acquisition. The amount of reduction compared to a full acquisition will depend on the specifics of each case. If all DataNodes in the cluster are in use and their disk filled with data potentially affected by the attack, the reduction in Phases 6 – 8 would be approx. 2/3 where the HDFS cluster uses the default replication factor of three (as our method ideally acquires a single replica of each of the potentially compromised blocks).

The size of the cluster per se should not directly affect the time and space requirements of the methodology. However, larger clusters are likely to be supported by a NameNode with larger RAM and to incur more transactions, increasing the size of the log files, which in turn will scale up the NameNode acquisition and analysis requirements. The DataNode acquisition and analysis requirements scale directly with the amount of data suspected to be affected by the attack if sufficient block offsets can be found to support partial selective acquisition from the relevant NameNodes. Otherwise, if entire NameNodes need to be used, the requirements scale with their typical disk size and the number of NameNodes the affected data is distributed across.

The fact that startOffset values were not able to be consistently retrieved supports the recommendations of Spyridopoulos and Katos (2011) that technology providers should implement a persistent offset-to-block mapping in the NameNode's storage. Given that the NameNode already stores persistent file system metadata in its FsImage file, the inclusion of this information populated from the NameNode's RAM could feasibly be handled with minimum performance impact. Such an addition would undoubtedly increase the forensic readiness of the Hadoop architecture and speed up forensic procedure following an attack.

8.2 Evaluation

This section evaluates our method against the criteria of the NIST Computer Forensic Tool Testing (CFTT) Project, which specifies completeness and accuracy criteria for computer forensic tools (NIST, 2004). Any future tool designed to automate our methodology must be validated against the CFTT criteria. To afford this, our methodology should also comply in order to provide a suitable foundation. The remainder of this section thus evaluates our method against the first six of the eight mandatory CFTT criteria. The remaining two are applicable only to automated tools as they refer to dealing with errors.

DI-RM-01: 'The tool shall be able to acquire a digital source using each access interface visible to the tool.'

In our methodology three areas for acquisition were identified: (a) the NameNode's RAM, (b) the NameNode's persistent volatile artefacts (Edits log and FsImage file) and (c) images of targeted DataNodes, ideally selective of the specific blocks pinpointed. Due to the highly differential nature of these areas of acquisition, it is improbable that a single forensic tool would seek to achieve all three phases via the same visual interface. A wealth of disk imaging tools exist which comfortably satisfy the requirements of imaging a DataNode. For example, the dd tool used in our case study and the related dcfldd could be used for targeted partial acquisition if the required offsets have been found, or for acquisition of an entire DataNode where this is not the case. Given that DataNodes are designed to function on commodity hardware, it is unlikely that the need for a specialist acquisition tool to this end should arise. Conversely, an access interface tool specially intended for NameNode acquisition is an area of future research that warrants attention. No such tool is listed in the most recent NIST CFTT handbook (NIST, 2015), nor indeed are any tools specifically designed for acquisition within HDFS.

DI-RM-02: 'The tool shall be able to create either a clone of a digital source, or an image of a digital source, or provide the capability for the user to select and then create either a clone or an image of a digital source.'

With knowledge of the local operating system implemented by the NameNode, a tool based on our methodology could meet this requirement by imaging the digital sources identified, namely the RAM and log files, and provide a suitable visual interface.

DI-RM-03: 'The tool shall operate in at least one execution environment and shall be able to acquire digital sources in each execution environment.'

Any tool created specifically for Hadoop forensics must be compatible with HDFS as a basic and fundamental requirement. As HDFS is built for Unix based operating systems, the tool should be compatible with a number of Unix based operating systems, and agnostic to the native file systems and its endian structure. As our methodology uses HDFS and Unix based commands, it provides a suitable foundation. A further consideration is that companies such as Cloudera, Hortonworks and MapR offer managed proprietary Hadoop distributions complete with their own inbuilt administrative command line interfaces. A forensic tool should be compatible with the target HDFS system regardless of whether it is implemented through a proprietary provider or not. Finally, Hadoop's backend functionality has changed substantially between 1.x.x and 2.x.x versions, particularly in regards to checkpointing and job scheduling. This complicates the development of a tool, as it should be applicable to all existing Hadoop series and multiple versions.

DI-RM-04: 'The tool shall completely acquire all visible data sectors from the digital source.'

As discussed in relation to DI-RM-02, the data sectors targeted for acquisition on the NameNode are volatile. Whilst our methodology was able to capture these completely, the analysis presented in Section 7.1 found that the crucial 'startOffset' elements were not always present in the RAM capture relating to a browseBlock.jsp process, even when corresponding HDFS block IDs had been identified in the same process in RAM. Our case study has demonstrated that full compliance with this condition would rely on permanent storage of physical block locations on the NameNode. We concur with

Spyridopoulos and Katos (2011) in recommending that such permanent storage should be implemented by developers of distributed file systems in order to improve forensic readiness. In the absence of this, a tool would have to acquire the identified subset of DataNodes in their entirety, which would present the issues of privacy due to multi-tenancy and data volume discussed earlier.

DI-RM-05: 'The tool shall completely acquire all hidden data sectors from the digital source.'

'Hidden data sectors' refers to host protected areas within the target system which cannot be accessed through standard file system read/write operations. This is relevant to a tool based on our methodology as the acquisition of the /dev/mem RAM repository requires kernel level access which is not normally available (see Section 6.2, Phase 2). Fmem was shown to be suitable for RAM capture in our case study as it installs a kernel module on the target system which circumvents Linux's native protection. This could be used by an automated acquisition tool.

DI-RM-06: 'All data sectors acquired by the tool from the digital source shall be accurately acquired.'

Our methodology sets as one of its core requirements the need to minimise direct cluster interaction. An automated acquisition tool based on the method should therefore be an accurate representation of the RAM image at the time of acquisition, though its accurate representation of the RAM at the time of the attack would depend on the operations logged during the elapsed time. Further to this, acquiring a copy of the FsImage and Edits log and performing a checkpoint operation via a forensic workstation will ensure that the examiner has access to both the up to date HDFS namespace (once all transaction have been mapped), and all the transactions in the Edits log that occurred between the previous checkpoint and the attack.

9. Conclusion

This paper has provided a suggested methodology for Hadoop forensics, tested the NameNode RAM acquisition and analysis phases of the methodology against a live Hadoop cluster in a case study and discussed the methodology's compliance with the NIST CFTT framework. Future work should be undertaken to test the methodology against larger cluster implementations, and further to extend the experience to include forensic analysis of the targeted DataNodes. While our scenario of deleted data presents the 'worst case' (see Sections 4 and 6.1), the methodology should also be tested against scenarios of data corruption or modification, and situations where it is unknown how the data was compromised.

Further to this, the methodology in its current state is applicable to in-house Hadoop scenarios. The greater complexity of outsourced environments means that in such instances Hadoop forensics will remain a problematic area. This is exemplified by the fact that the Linux operating system requires the installation of a kernel module in order to perform RAM acquisition (Section 6.2 Phase 2).

When factoring in issues beyond technicality (i.e. multi-tenancy, and physical cluster access) it is clear that we are a long way from complete forensic readiness in outsourced Hadoop implementations. Further work needs to be conducted academically and collaboratively with technology providers to address these issues, and provide a workable solution to outsourced Hadoop implementations.

The aim of this project was to simulate a data breach attack within a Hadoop cluster, and design and implement a suitable methodology for forensic investigation thereof. Our methodology is drawn from existing research in the area, and utilises an initial live acquisition stage on the NameNode. This predicates further targeted acquisition on the DataNodes. The rationale behind this methodology being that with awareness of what data blocks were affected by an attack a move away from the dated blanket acquisition approach of traditional forensics could be sought, which given the scale of Hadoop implementations is necessary for future forensic approaches. By identifying two strings that can be used as magic number during the RAM analysis on the NameNode, the investigation was able to establish a proportion of blockIDs and startOffset values relating to deleted HDFS data blocks, although complete transparency was not available. This supports the suggestion of the designed methodology

to include NameNode RAM forensics as a prerequisite to Hadoop forensics, as the identification of the deleted blockIDs in the namespace would enable the hosting DataNodes to be targeted for imaging.

Acknowledgements

Some of the practical work described in this paper was carried out at and supported by Abertay University, Dundee, United Kingdom in the context of a student Honours Project.

References

Apache. Hadoop 1.2.1 documentation. 2013. [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/ [accessed 19/04/2015].

Apache. HDFS User guide version 2.4.1. 2014. [Online]. Available: https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html [accessed 31/10/15].

Carrier B. The Sleuth kit Hadoop framework. 2013. [Online]. Available: http://www.sleuthkit.org/tsk_hadoop/index.php [accessed 20/10/15]

Cho CH, Chin SH, Chung KS. Cyber forensic for Hadoop based cloud system. Int J Secur Its Appl 2012; 6(3):83-90.

Cloudera. Configuring HDFS trash. 2015. [Online]. Available: http://www.cloudera.com/documentation/enterprise/5-4-x/topics/cm_mc_config_trash.html [accessed 03/03/2016]

Daryabar F, Dehghantanha A, Udzir NI. A survey about impacts of cloud computing on digital forensics. Int J Cybersecur Digit Forensics 2013; 2(2):77-94.

Fowler K. Hadoop forensics: Tackling the elephant in the room. SecTor. 2012. [Online]. Available: http://2012.video.sector.ca/video/51118145 [accessed 19/04/2015]

Garfinkel S. Digital forensics research: The next 10 years. Digit Investig 2010; 7:64-73.

Grispos G, Storer T, Glisson WB. Calm before the storm: The challenges of cloud computing in digital forensics. Int J Digit Crime Forensics. 2012; 4(2):28-48.

Hegarty R, Merabti M, Shi Q, Askwith B. Forensic analysis of distributed data in a service oriented computing platform. In: Clarke N, Tryfonas T, Dodge R, eds. Proc 7th Int Workshop Digit Forensics. Plymouth: University of Plymouth, 2012:27-37.

ISC^2. Security in the skies: cloud computing security concerns, threats, and controls. White Paper IX. 2013. [Online]. Available: https://www.isc2.org/csslp-whitepaper.aspx. [accessed 31/10/2015].

Lallie HS, Pimlott L. Challenges in applying the ACPO Principles in cloud forensic environments. J Digit Forensics, Security and Law, 2012; 7(1):71-85.

Lee J, Hong D. Pervasive forensic analysis based on mobile cloud computing. Int Conf Multimedia Information Networking and Security, Shanghai Nov2011; 572-76.

Lee TR, Lee H, Rhee KH, Shin SU. The efficient implementation of distributed indexing with hadoop for digital investigations on big data. Comput Sci Inf Syst 2014; 11(3):1037-54.

Lee J, Un S. Digital forensics as a service: A case study of forensic indexed search. In: ICT Convergence (ICTC), Int Conf. Jeju Island, Korea Oct2012; 499–503.

Martini B, Choo K-KR. An integrated conceptual digital forensic framework for cloud computing. Digit Investig 2012; 9:71-80.

Martini B, Choo K-KR. Distributed filesystem forensics: XtreemFS as a case study. Digit Investig 2014a; 11:295-313.

Martini B, Choo K-KR. Remote programmatic vCloud forensics: A six-step collection process and a proof

of concept. IEEE 13th Int Conf on Trust, Security and Privacy in Comput and Comm 2014b; 935-942.

NIST (National Institute of Standards and Technology). Computer forensics tool testing handbook. 2015. [Online]. Available: http://www.cftt.nist.gov/CFTT-Booklet-08112015.pdf [accessed 03/03/2016]

NIST. Digital data acquisition tool specification. 2004. [Online]. Available: http://www.cftt.nist.gov/Pub-Draft-1-DDA-Require.pdf [accessed 03/03/2016]

Patrascu A, Patriciu V. Logging framework for cloud computing forensic environments. Communications (COMM), 10th Int Conf 2014; 1-4.

Poisel R, Malzer E, Tjoa S. Evidence and cloud computing: The virtual machine introspection approach. J Wirel Mobile Networks, Ubiquitous Comput Dependable Appl. 2013; 4(1):135-52.

Quick D, Choo K-KR. Impacts of increasing volume of digital forensic data: A survey and future research challenges. Digit Invest 2014; 11:273-94.

Ruan K, Carthy J, Kechadi T, Crosbie M. Cloud Forensics. In: Peterson G. and Shenoi, S. (eds). Advances in Digital Forensics VII: Springer IFIP Advances in Information and Communication Technology 2011; 361:35-46.

Spyridopoulos T and Katos V. Requirements for a forensically ready cloud storage service. Int J Digit Crime and Forensics 2011: 3(3):19-36.

Ubuntu. Security features. 2015. [Online]. Available: https://wiki.ubuntu.com/Security/Features [accessed 12/04/15].

Wang A. Guide to checkpointing in Hadoop. 2014. [Online]. Available: http://blog.cloudera.com/blog/2014/03/a-guide-to-checkpointing-in-hadoop/ [accessed 20/11/2014]

White T. Hadoop: The definitive guide. 4th ed. Sebastopol, CA: O'Reilly; 2014.