# A Forensic Analysis Method for Redis Database Based on RDB and AOF File

Ming Xu*
College of Computer, Hangzhou Dianzi University, Hangzhou, China
Email: mxu@hdu.edu.cn

Xiaowei Xu, Jian Xu, Yizhi Ren, Haiping Zhang, Ning Zheng
College of Computer, Hangzhou Dianzi University, Hangzhou, China
Email: soap8815@gmail.com, { jian.xu, renyz, zhanghp, nzheng }@hdu.edu.cn

***Abstract*—Redis is a widely used non-relational and in-memory database system. It holds a large amount of information both in memory and file system, which is of great significance to forensic analysis. This paper mainly proposes a forensic analysis method for Redis based on RDB and AOF file. A method of extracting useful information from RDB backup file is proposed based on the data storage mechanism described in this paper. A method of reconstructing the write operation statements from AOF file is also provided. Finally, the method of directly analyzing data from memory is shown. The experimental results demonstrate the effectiveness of our method. Most of the data could be extracted from RDB and AOF file, which provides important information for forensic investigators.**

***Index Terms*—Redis, NoSQL, database forensics, digital forensics**

## I. INTRODUCTION

Database systems play an important role in every aspect of our life. They typically hold massive amount of data and form the basis for various applications [1-2]. Relational databases work quite well when the data stored in them is highly structured with strict relations between them [3]. However, a lot of applications today use data structures like lists, sets, hashes, and graphs. Storing these less structured data into traditional relational databases would require complex mapping algorithm and often lead to poor performance. In addition to this, if the data set is too large to fit into one server, the database should be partitioned into multiple servers, which is a weak point of many relational databases due to their complex deployment and bad performance. In general, if we are dealing with large amounts of data or the data is less structured, we might have better options than relational databases.

NoSQL databases provide an alternative way to store data other than relational databases [4]. NoSQL databases are most useful when working with a huge quantity of data that does not require a relational model. Aside from being non-relational, most NoSQL databases are also distributed, open-source and horizontally scalable. There are approximately 150 different NoSQL databases. They could be roughly divided into four categories: document store databases, key-value store databases, graph databases and BigTable Column Family Store databases. Among them, key-value store databases have the simplest form for storing data. Each key is mapped to a value containing arbitrary data. Redis is the most widely used key-value store database and is rapidly gaining popularity all across the globe [5].

Redis forensics is of great importance in many aspects. First, Redis is widely used in many companies to store large amount of data and would thus be a primary target in a forensic investigation. Second, some data in Redis might be mistakenly removed by database users. Redis forensics provides a way to recover these deleted data. Third, Redis is a potential target of database intrusions that involves stealing or tampering the database data. The data recovered in Redis could be used to prove a database security breach and determine the scope of a database intrusion. Finally, the study on Redis forensics could help the study on some other NoSQL databases with similar key-value storage mechanism like Riak and Cassandra. The study on Redis forensics also provides insight into the forensic techniques of some other memory databases. We could analyze the disk backup file instead of the memory to extract the database data.

Both the RDB file and AOF file are of great forensic value in Redis forensics. First, while extracting and analyzing data from memory directly is very difficult, it is relatively easy to parse the RDB file and AOF file instead and extract the data. Second, some deleted data in memory might still be found in RDB file. Third, the RDB file and AOF file could be used to recover important data when the Redis server crashes and the data in memory is lost. Last but not least, by examining the data extracted from AOF file, we could learn what write operations are performed in Redis.

The goal of this paper is to show how to extract data from the Redis RDB backup file. A method to parse the AOF log file and reconstruct write operation statements is also proposed. We also briefly explain how the data is stored in memory.

In Section 2, we provide a brief overview of related work in the field of database forensics. In Section 3, we


doi:10.4304/jcp.9.11.2538-2544

describe the structure of Redis RDB and AOF file. Section 4 shows our algorithms to extract data from these files and Section 5 discusses the corresponding experiment. We also briefly cover the topic of Redis memory forensics in Section 6. We conclude our work in Section 7.

## II. Related Work

Database forensics is a very important research field that has received little research attentions these years. Martin S Olivier [6] believed the lack of research is due to the inherent complexity of databases that is not fully understood in a forensic context. Harmeet Kaur Khanuja [7] discussed various methodologies for tamper detection in databases and outlined challenges and opportunities in database forensics. He [8] also proposed a framework that builds the expert system for database analysis in two stages. Patrick Stahlberg [9] demonstrated that existing database systems fail to securely remove deleted data and remnants of past operations, making the recovery of deleted data possible.

Peter Frühwirt [10-12] described the file format of the MySQL Database with InnoDB Storage Engine and proposed methods for recovering basic SQL statements by analyzing InnoDB's redo logs. Paul M. Wright [13] introduced advanced Oracle forensics techniques, which could ensure the safety and security of Oracle data. David Litchfield [14-19] performed forensic analysis of a compromised Oracle database server and discussed every aspect of Oracle forensics in his series of papers. Kevvie Fowler [20] defined, established, and documented SQL server forensic methods and techniques in his book.

Josiah L. Carlson [21] introduces Redis and explains how to use Redis effectively in his book. Tiago Macedo and Fred Oliveira [22] provide recipes for a variety of issues a Redis user will face in their book.

## III. Redis Internals

In this section, we explain in great details the structure of Redis RDB file and AOF file, which forms the basis of our algorithms to extract useful data from these files.

### A. *RDB File Format*

By default, the whole Redis dataset resides in volatile memory. But Redis would also save the snapshots of all the data in memory to a RDB file on disk. When a Redis server starts, the RDB file would be loaded into memory. RDB file is useful for purposes like backup and disaster recovery. Database users can copy RDB files to other machines and data centers even when the database is still running. By analyzing the RDB backup file, forensic investigators could extract the Redis data without the need to analyze the data in memory, which is fairly difficult.

#### *1). Overall File Structure*

We will first look at the overall structure of a RDB file. Table 1 shows the RDB file after two strings are inserted. Table 2 explains the meaning of the bytes.

TABLE I.
HEXADECIMAL STRUCTURE OF RDB FILE

| | | |
|---|---|---|
| 0x00000000 | 52 45 44 49 53 30 30 30 36 FE 00 FC 9C 45 6D 60 | REDIS0006……. |
| 0x00000010 | 3F 01 00 00 00 03 61 67 65 C0 14 00 04 6E 61 6D | ……age….nam |
| 0x00000020 | 65 04 4A 6F 68 6E FF 43 70 8B 5D AB 68 4A 84 | e.John……… |

TABLE II.
MEANING OF THE HEXADECIMAL VALUE OF THE RDB FILE

| Offset | Length | Value | Meaning |
|---|---|---|---|
| 0 | 5 | 52 45 44 49 53 | Magic number: Redis |
| 5 | 4 | 30 30 30 36 | Redis RDB Version: 6 |
| 9 | 2 | FE 00 | Database number:0 |
| 11 | 27 | FC 9C 45 6D 60 3F 01 00 00 00 03 61 67 65 C0 14 00 04 6E 61 6D 65 04 4A 6F 68 6E | Database data, stored as key-value pairs |
| 38 | 1 | FF | End of File marker |
| 39 | 8 | 43 70 8B 5D AB 68 4A 84 | Checksum |

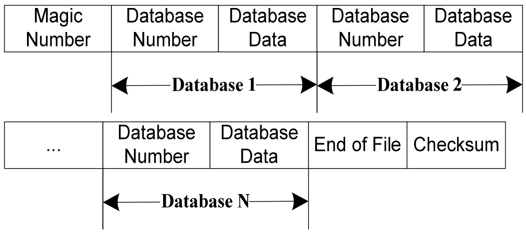


Figure 1. Overall Structure of a RDB File

Redis stores data in a database in the form of key-value pairs. Table 3 shows the meaning of the bytes in a key-value pair in Table 1.

TABLE IIII.
MEANING OF THE HEXADECIMAL VALUE OF A KEY-VALUE PAIR

| Offset | Length | Value | Meaning |
|---|---|---|---|
| 11 | 9 | FC 9C 45 6D 60 3F 01 00 00 | Expire time: 30 seconds |
| 20 | 1 | 00 | Type of value: 0 |
| 21 | 4 | 03 61 67 65 | Key: age |
| 25 | 2 | C0 14 | Value: 20 |

In Redis, each key could be associated to an expire time field, and will be removed automatically by Redis server when the specified amount of time has elapsed. The expire time is stored as an absolute Unix timestamps in milliseconds. The type of value field tells us which encoding method Redis uses in order to store the value field. The key field is a Redis string, and the value field is stored based on the encoding method described in the type of value field.

Redis is known for its rich support of various data structures. The value field in a key-value pair could be one of five data structures in Redis, which greatly facilitate the work of programmers. In the next few subsections, we will discuss how these data structures are stored in a RDB file.

#### *2). String*

String is the most common data structure in Redis. Currently, there are three ways to store a string in a RDB file. If the string is an 8, 16 or 32 bit integer, it can be stored as an integer. If the length of the string is greater than 20 and the LZF compression is enabled, the string would be compressed before being stored. In other cases, it would be stored simply as a byte array.

Table 4 shows the RDB file after three strings are inserted. They are stored in three different ways. Table 5 explains the meaning of the bytes.

TABLE III.
HEXADECIMAL STRUCTURE OF REDIS STRING

| | | |
|---|---|---|
| 0x00000000 | 52 45 44 49 53 30 30 30 36 FE 00 00 03 61 67 65 | REDIS0006….age |
| 0x00000010 | C0 14 00 04 6E 61 6D 65 04 4A 6F 68 6E 00 0B 64 | ….name.John..d |
| 0x00000020 | 65 73 63 72 69 70 74 69 6F 6E C3 0E 21 01 61 61 | escription…… |
| 0x00000030 | E0 06 00 00 62 E0 04 00 01 62 62 FF BA 80 4B AF | ……………. |
| 0x00000040 | DE 2B 67 09 | …. |

TABLE V.
MEANING OF THE HEXADECIMAL VALUE OF REDIS STRING

| Offset | Length | Value | Meaning |
|---|---|---|---|
| 11 | 1 | 00 | Type of value: 0. It indicates the value field stores a string. |
| 16 | 1 | C0 | Length Encoding. It represents the string is stored as an 8-bit integer |
| 17 | 1 | 14 | The value of the string: 20 |
| 18 | 1 | 00 | Type of value: 0. It indicates the value field stores a string. |
| 24 | 1 | 04 | Length Encoding. It represents the string is stored as a byte array and the length of the byte array is 4. |
| 25 | 4 | 4A 6F 68 6E | The content of byte array |
| 29 | 1 | 00 | Type of value: 0. It indicates the value field stores a string. |
| 42 | 1 | C3 | Length Encoding. It represents the string is an LZF compressed string |
| 43 | 1 | 0E | The length of the string after compression |
| 44 | 1 | 21 | The length of the string before compression |
| 45 | 14 | 01 61 61 E0 06 00 00 62 E0 04 00 01 62 62 | The content of the compressed string |

The aim of the length encoding field is to indicate how Redis stores the following string structure [23]. In default, length encoding field is one byte long and the two most significant bits show the way the string is stored. If the two bits are 00, 01 or 10, then the string is stored as a byte array. The length of the array is indicated in the following 6 bits if the starting two bits are 00. If the starting two bits are 01, the length of the array depends on the following 6 bits and the next byte combined, which is 14 bits in total. If the starting two bits are 10, the next 4 bytes represent the length. If the starting two bits are 11 and the value of the remaining 6 bits is 0, 1 or 2, it represents the string is stored as an 8-bit, 16-bit or 32-bit integer, respectively. If the starting two bits are 11 and the value of the remaining 6 bits is 4, it means the string is compressed. Redis adopts LZF compression algorithm to store a long string.

*3) List*

In Redis, list is a collection of strings sorted by insertion order. Lists are widely used in various situations, including modeling a timeline and passing messages [24]. By default, Redis stores a list as a ziplist. But when the number of elements in a list exceeds *server.list_max_ziplist_entries* or the length of any of these elements exceeds *server.list_max_ziplist_value*, the list would be stored as a linkedlist alternatively. *server.list_max_ziplist_entries* and *server.list_max_ziplist_value* are predefined and configurable values in Redis.

Table 6 shows the RDB file after two lists are inserted, one with two elements stored as a ziplist and one with one element stored as a linkedlist. Table 7 explains the meaning of the bytes.

TABLE VI.
HEXADECIMAL STRUCTURE OF REDIS LIST

| | | |
|---|---|---|
| 0x00000000 | 52 45 44 49 53 30 30 30 36 FE 00 0A 0A 70 61 72 | REDIS0006….par |
| 0x00000010 | 61 6D 65 74 65 72 73 13 13 00 00 00 0D 00 00 00 | ammeters……… |
| 0x00000020 | 02 00 00 FE 14 03 03 63 61 72 FF 01 07 73 74 72 | …….car...str |
| 0x00000030 | 69 6E 67 73 01 C3 1D 40 C6 01 61 61 E0 19 00 00 | ings………… |
| 0x00000040 | 62 E0 24 00 00 63 E0 22 00 00 64 E0 1E 00 00 65 | ……………. |
| 0x00000050 | E0 14 00 01 65 65 FF 80 9B C2 3E 59 11 38 12 | …………… |

TABLE VII.
MEANING OF THE HEXADECIMAL VALUE OF REDIS LIST

| Offset | Length | Value | Meaning |
|---|---|---|---|
| 11 | 1 | 0A | Type of value: 10. It indicates the value field is a ziplist. |
| 24 | 4 | 13 00 00 00 | The number of bytes the ziplist occupies : 19. |
| 28 | 4 | 0D 00 00 00 | Offset to the last element in the ziplist: 13. |
| 32 | 2 | 02 00 | The number of elements in the ziplist: 2. |
| 34 | 1 | 00 | The number of bytes the previous element occupies: 0. |
| 35 | 1 | FE | Length Encoding. It represents the element is stored as an integer |
| 36 | 1 | 14 | The value of the first element: 20. |
| 37 | 1 | 03 | The number of bytes the previous element occupies: 3. |
| 38 | 1 | 03 | Length Encoding. It represents the element is stored as a byte array and the length of the array is 3 |
| 39 | 3 | 63 61 72 | The value of the second element. |
| 42 | 1 | FF | End of ziplist marker |
| 43 | 1 | 01 | Type of value: 1. It indicates the value field is a linkedlist. |
| 52 | 1 | 01 | The number of elements in the linkedlist:1 |
| 53 | 33 | C3 1D 40 C6 01 61 61 E0 19 00 00 62 E0 24 00 00 63 E0 22 00 00 64 E0 1E 00 00 65 E0 14 00 01 65 65 | The content of the first element: an LZF compressed string. |

| Ziplist Length | Offset to the Last Element | The Number of Elements | Element 1 |
|---|---|---|---|
| Element 2 | ... | Element N | End of Ziplist |

Figure 2. Structure of a Ziplist

| Length of the Previous Element | Length Encoding | Data |
|---|---|---|

Figure 3. Structure of a Ziplist Element

| The Number of Elements | Element 1 | Element 2 | ... | Element N |
|---|---|---|---|---|

Figure 4. Structure of a Linkedlist

*4). Set*

Set in Redis is a collection of unordered strings. Sets have the property of not allowing repeated elements, which makes set an ideal structure for tracking unique things and representing relations. When all the elements in a set are integers and the number of elements does not exceed *server.set_max_intset_entries*, the set would be stored as an intset. If either of these two conditions is not met, Redis would use a hash table to store the set.

Table 8 shows the RDB file after two sets are inserted, each with two elements. The first set is stored as a hash table and the second is stored as an intset. Table 9 explains the meaning of the bytes.

TABLE VIII.
HEXADECIMAL STRUCTURE OF REDIS SET

| | | |
|---|---|---|
| 0x00000000 | 52 45 44 49 53 30 30 30 36 FE 00 02 06 61 6E 69 | REDIS0006….ani |
| 0x00000010 | 6D 61 6C 02 05 73 6E 61 6B 65 05 74 69 67 65 72 | mal..snake.tiger |
| 0x00000020 | 0B 05 73 63 6F 72 65 0C 02 00 00 00 02 00 00 00 | ..score……… |
| 0x00000030 | 50 00 5F 00 FF F4 9B 09 2D DD 38 FC 61 | ………… |

TABLE IX.
MEANING OF THE HEXADECIMAL VALUE OF REDIS SET

| Offset | Length | Value | Meaning |
|---|---|---|---|
| 11 | 1 | 02 | Type of value: 2. It indicates the value field is a hash table. |
| 19 | 1 | 02 | The number of elements in the hash table:2 |
| 20 | 1 | 05 | The number of bytes the first element occupies. |
| 21 | 5 | 73 6E 61 6B 65 | The content of the first element |
| 26 | 1 | 05 | The number of bytes the second element occupies. |
| 27 | 5 | 74 69 67 65 72 | The content of the second element |
| 32 | 1 | 0B | Type of value: 11. It indicates the value field is an intset. |
| 39 | 1 | 0C | The number of bytes the intset occupies: 12. |
| 40 | 4 | 02 00 00 00 | Encoding of element in the intset: 2. It means each element is stored as a 2-byte integer |
| 44 | 4 | 02 00 00 00 | The number of elements in the intset |
| 48 | 2 | 50 00 | The content of the first element |
| 50 | 2 | 5F 00 | The content of the second element |

| Intset Length | Encoding of Elements | The Number of Elements | Element 1 | Element 2 | ... | Element N |
|---|---|---|---|---|---|---|

Figure 5. Structure of an Intset

| The Number of Elements | Element 1 | Element 2 | ... | Element N |
|---|---|---|---|---|

Figure 6. Structure of a Hash Table

*5) Sorted Set*

In addition to sets, Redis also provides sorted sets. The main difference between these two data structures is that each element in a sorted set has two parts: *member* and *score*. The sorted set is ordered based on the *score* field and the *member* field stores the actual data. You can do a lot of tasks with sorted sets, like indexing data and making a leader board. If the number of elements in a sorted set is smaller than *server.zset_max_ziplist_entries* and there is no element with a member whose length exceeds *server.zset_max_ziplist_value*, it would be stored using ziplist. Otherwise, a skiplist is used to store the sorted list.

Table 10 shows the RDB file after two sorted sets are inserted, one with two elements stored as a skiplist and one with three elements stored as a ziplist. Table 11 explains the meaning of the bytes.

TABLE X.
HEXADECIMAL STRUCTURE OF REDIS SORTED SET

| | | |
|---|---|---|
| 0x00000000 | 52 45 44 49 53 30 30 30 36 FE 00 03 04 77 6F 72 | REDIS0006….wor |
| 0x00000010 | 64 02 C3 13 40 BF 01 66 66 E0 38 00 00 67 E0 3B | ……………. |
| 0x00000020 | 00 00 6A E0 2B 00 01 6A 6A 01 32 C3 13 40 A0 01 | ……………. |
| 0x00000030 | 63 63 E0 2C 00 00 64 E0 30 00 00 65 E0 23 00 01 | ……………. |
| 0x00000040 | 65 65 01 31 0C 05 63 6F 6C 6F 72 23 23 00 00 00 | ……………. |
| 0x00000050 | 20 00 00 00 06 00 00 04 62 6C 75 65 06 F2 02 05 | ……..blue…. |
| 0x00000060 | 67 72 65 65 6E 07 F3 02 03 72 65 64 05 F4 FF FF | green….red…. |
| 0x00000070 | 80 98 1B CB 8B 4C 05 32 | …….. |

TABLE XI.
MEANING OF THE HEXADECIMAL VALUE OF REDIS SORTED SET

| Offset | Length | Value | Meaning |
|---|---|---|---|
| 11 | 1 | 03 | Type of value: 3. It indicates the value field is a skiplist |
| 17 | 1 | 02 | The number of elements in the skiplist: 2. |
| 18 | 23 | C3 13 40 BF 01 66 66 E0 38 00 00 67 E0 3B 00 00 6A E0 2B 00 01 6A 6A | The member field of the first element |
| 41 | 2 | 01 32 | The score field of the first element |
| 43 | 23 | C3 13 40 A0 01 63 63 E0 2C 00 00 64 E0 30 00 00 65 E0 23 00 01 65 65 | The member field of the second element |
| 66 | 2 | 01 31 | The score field of the second element |
| 68 | 1 | 0C | Type of value: 12. It indicates the value field is a ziplist. It contains 3 pairs of member and score. The structure of ziplist has been discussed above. We won't go into details here. |


Figure 7. Structure of a Skiplist

### 6) *Hash*

Hash in Redis is a collection of key-value pairs. They are primarily used to represent objects. If the number of element in a hash is less than *server.hash_max_ziplist_entries* and all the keys and values are of smaller length than *server.hash_max_ziplist_value*, it would be stored as a ziplist. In all other cases, a hash table will be used to store the hash.

Table 12 shows the RDB file after two hashes are inserted, one with two key-value pairs stored as a ziplist and one with one key-value pair stored as a hash table. Table 13 explains the meaning of the bytes.

TABLE XII.
HEXADECIMAL STRUCTURE OF REDIS HASH

| | | |
|---|---|---|
| 0x00000000 | 52 45 44 49 53 30 30 30 36 FE 00 04 07 61 72 74 | REDIS0006….art |
| 0x00000010 | 69 63 6C 65 01 05 74 69 74 6C 65 C3 13 40 9C 01 | Icle..title….. |
| 0x00000020 | 68 68 E0 33 00 00 69 E0 28 00 00 6A E0 20 00 01 | ……………. |
| 0x00000030 | 6A 6A 0D 06 70 65 6F 70 6C 65 1F 1F 00 00 00 1B | ….people…… |
| 0x00000040 | 00 00 00 04 00 00 04 6E 61 6D 65 06 04 4A 61 63 | …….name..Jac |
| 0x00000050 | 6B 06 03 61 67 65 05 FE 1E FF FF AA 08 88 C2 DF | …age………. |
| 0x00000060 | 68 6F 8C | … |

TABLE XIII.
MEANING OF THE HEXADECIMAL VALUE OF REDIS HASH

| Offset | Length | Value | Meaning |
|---|---|---|---|
| 11 | 1 | 04 | Type of value: 4. It indicates the value field is a hash table |
| 20 | 1 | 01 | The number of key-value pairs in the hash table: 1. |
| 21 | 6 | 05 74 69 74 6C 65 | Key of the first key-value pair |
| 27 | 23 | C3 13 40 9C 01 68 68 E0 33 00 00 69 E0 28 00 00 6A E0 20 00 01 6A 6A | Value of the first key-value pair |
| 50 | 1 | 0D | Type of value: 13. It indicates the value field is a ziplist. It stores 2 key-value pairs. The structure of ziplist has been discussed above. We won't go into details here. |

## B. *AOF File Format*

Apart from RDB, Redis also provides AOF as another data persistence strategy. Instead of writing all the data in the memory to the disk, AOF simply logs every write operation. To prevent the AOF file from getting too big, Redis supports the feature of rebuilding the whole AOF file by writing the shortest sequence of statements needed to build the current database in memory. As a result, the AOF file may not show all the write operations performed by the database user, but it is still a good source from which to reconstruct the write operation statements.

The AOF file is a text file and features a relatively simple data format. Every write operation in Redis would be recorded in the AOF file as a single record. Figure 8 describes the format of a record.

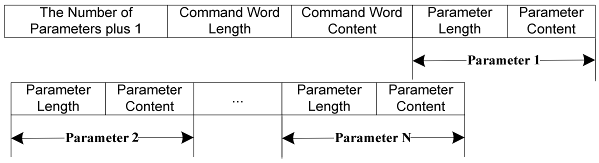


Figure 8. Structure of a record in an AOF file

# IV. THE PROPOSED METHOD

## A. *Extract Data from RDB File*

Data is extracted from RDB file based on Algorithm 1. After extracting the magic number and RDB file version, we extract the data in each database. Extracting the Redis key is simple because the key is normally stored as a string. But since the Redis value is of various data structures and storing methods, it requires different methods to extract them. For some simple structures like linkedlist and hash table, we could simply iterate through all the elements; while for structures like ziplist, we could iterate through all the elements in reverse order. To extract data from intset, we need to first determine the encoding method of the elements.

ALGORITHM I
Extract data from RDB file

**Algorithm 1**: Extract data from RDB file

**Input**: *file*: a Redis RDB backup file

**Output**: *data structures*: a list of Redis data structures extracted from the RDB file

```
i = 0
extract the magic number and RDB version
while i < file.length do    // scan the entire file
    Case file[i] of
        Case FE:
            extract the database number
        Case 0:   // the value is a string
            extract the key
            extract the value based on length encoding
        Case 10:   // the value is a list stored as a ziplist
            extract the key
            extract all the elements in the ziplist based on the offset and number of elements fields
        Case 11:   // the value is a set stored as an intset
            extract the key
            extract all the elements in the intset based on the encoding of elements and number of elements fields
        Case 2:   // the value is a set stored as a hash table
            extract the key
            extract all the elements in the hash table based on the number of elements field
        Case 12:   // the value is a sorted set stored as a ziplist
            extract the key
            extract all the elements in the ziplist based on the offset and number of elements fields
        Case 13:   // the value is a hash stored as a ziplist
            extract the key
            extract all the elements in the ziplist based on the offset and number of elements fields
    endcase
endwhile
```

## B. *Reconstruct Write Operation Statements from AOF File*

Write operation statements are extracted and reconstructed from AOF file based on algorithm 2. For each write operation statement, we extract its statement content and all the statement parameters successively.

ALGORITHM II
Reconstruct write operation statements from AOF file

**Algorithm 2**: Reconstruct write operation statements from AOF file

**Input**: *file*: a Redis AOF file
**Output**: *statements*: a list of Redis write operation statements extracted from the AOF file

```
read a text line from file
while not EOF do
    if line[0] = '*' then
        mark the end of the last statement
        output a line break
        mark the end of a new statement
    elsif line[0] = '$' then
        ignore this line, continue the loop
    else
        output the line
    endif
    read the next line
endwhile
```

## V. Experiment And Evaluation

We implement an analysis tool using Java. The tool could analyze the input RDB and AOF file based on the algorithms described in section 4 and output the results to a text file.

To validate our method, we conduct experiments on a Linux machine. We first execute 100 write operation statements which insert 100 Redis data structures of various kinds with different data types into two empty Redis databases. After this, we run our analysis tool using the RDB file and AOF file extracted from the disk as input. We then analyze the output file.

Table 14 and Table 15 show the experimental result of extracting data from RDB file. High-level data structures are the structures that programmers could use when coding with Redis as underlying database. Redis server stores these high-level data structures in the disk using low-level data structures. For each data structure, we compare the number of total structures and the number of extracted structures. The experimental result shows that most of the data structures in the RDB file could be extracted using our method. The five structures that we fail to extract are all stored using LZF compression algorithm because the length of at least one of the elements in the structure exceeds a predefined threshold, which is not common in a real situation. Our method does not consider this situation.

TABLE XIV.
EXPERIMENTAL RESULT OF EXTRACTING DATA FROM RDB FILE(HIGH-LEVEL DATA STRUCTURES)

| | high-level data structures | | | | |
|---|---|---|---|---|---|
| | string | list | set | sorted set | hash |
| extracted structures | 20 | 18 | 20 | 18 | 19 |
| total structures | 20 | 20 | 20 | 20 | 20 |

TABLE XV.
EXPERIMENTAL RESULT OF EXTRACTING DATA FROM RDB FILE(LOW-LEVEL DATA STRUCTURES)

| | low-level data structures | | | | | |
|---|---|---|---|---|---|---|
| | string | ziplist | linkedlist | intset | skiplist | hash table |
| extracted structures | 20 | 45 | 3 | 10 | 3 | 14 |
| total structures | 20 | 45 | 5 | 10 | 5 | 15 |

Out of the 100 write operation statements we execute, our tool succeeds in extracting and reconstructing all of these statements from Redis AOF file.

These experimental results demonstrate that our method is effective in extracting data from the disk file, which is of great value both to forensic investigators and database users. Currently, our method cannot extract structure with compressed string in it. And our tool could not work when the file is corrupted.

## VI. Redis Memory Forensics

Redis is an in-memory database. All the data resides in volatile memory while disk only stores backup and log files. This feature gives Redis a huge advantage in performance compared to traditional disk databases. Redis can perform over 110000 write operations and 81000 read operations each second.

Although analyzing and extracting data directly from memory is no easy task, we will give a brief overview of how data in Redis is distributed in memory.

We use the fmem module to obtain a Linux volatile memory image. Fmem is a tool which could gain directly access to Linux physical memory and copy the whole memory image to the disk. After examining the memory image, we discover that some data structures have the same format both in memory and in RDB file, while others have different format. We also find multiple copies of write operation statements we executed all across the memory image, which is of some forensic value as well.

## VII. Conclusion

In this paper, we discussed the file format of Redis RDB backup file and AOF file, which makes it possible to extract data from these files. A prototype tool was implemented to verify our approach. Most of the data in the RDB and AOF file could be extracted using our tool. We further explained how data is distributed in memory.

This paper is among the first to explore the field of NoSQL database forensics and will prove to be valuable both in understanding the Redis internal architecture and recovering information from the database. Furthermore, our findings would also be helpful in understanding other memory databases and applications using similar data structures to store data.

Not all details of Redis data structures are covered in this paper and there is still broad room for further study of the behavior of data in memory. In the future, we plan to enhance the function of our forensic tool and dig deeper into Redis memory forensics.

## ACKNOWLEDGMENT

This work is supported by the Natural Science Foundation of China under Grant No.61070212 and 61003195, the Zhejiang Province Natural Science Foundation of China under Grant No.Y1090114 and LY12F02006, the Zhejiang Province key industrial projects in the priority themes of China under Grant No 2010C11050, the science and technology search planned projects of Zhejiang Province (No.2012C21040), the soft science research project of Hangzhou (No. 20130834M15).

## REFERENCES

[1] Jin Z, Xu Q. The Realization of Decision Support System for Cross-border Transportation based on the Multidimensional Database[J]. Journal of Software (1796217X), 2012, 7(5).

[2] Jin C, Liu N, Qi L. Research and Application of Data Archiving based on Oracle Dual Database Structure[J]. Journal of Software (1796217X), 2012, 7(4).

[3] Yang S, Shi L. Results Clustering for Keyword Search over Relational Database[J]. Journal of Software, 2013, 8(12): 3188-3193.

[4] Wikipedia NoSQL. http://en.wikipedia.org/wiki/NoSQL

[5] Wikipedia Redis http://en.wikipedia.org/wiki/Redis

[6] Olivier M S. On metadata context in Database Forensics[J]. digital investigation, 2009, 5(3): 115-123.

[7] Khanuja H K, Adane D D S. Database Security Threats and challenges in Database Forensic: A survey[C]//Proceedings of 2011 International Conference on Advancements in Information Technology (AIT 2011), available at http://www. ipcsit. com/vol20/33-ICAIT2011-A4072. pdf. 2011.

[8] Khanuja H K, Adane D D S. A Framework For Database Forensic Analysis[J]. Published in Computer Science & Engineering: An International Journal (CSEIJ), 2012, 2(3).

[9] Stahlberg P, Miklau G, Levine B N. Threats to privacy in the forensic analysis of database systems[C]//Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, 2007: 91-102.

[10] Fruhwirt P, Huber M, Mulazzani M, et al. InnoDB database forensics[C]//Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. IEEE, 2010: 1028-1036.

[11] Fruhwirt, P., Kieseberg, P., Schrittwieser, S., Huber, M., & Weippl, E. (2012, August). InnoDB database forensics: reconstructing data manipulation queries from redo logs. In Availability, Reliability and Security (ARES), 2012 Seventh International Conference on (pp. 625-633). IEEE.

[12] Fruhwirt, P., Huber, M., Mulazzani, M., & Weippl, E. R. (2010, April). InnoDB database forensics. In Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on (pp. 1028-1036). IEEE.

[13] Wright P M, Burleson D. Oracle Forensics: Oracle Security Best Practices[M]. Rampant TechPress, 2008.

[14] Litchfield D. Oracle Forensics Part 1: Dissecting the Redo Logs[J]. NGSSoftware Insight Security Research (NISR), Next Generation Security Software Ltd., Sutton, 2007.

[15] Litchfield D. Oracle forensics part 2: Locating dropped objects[J]. NGSSoftware Insight Security Research (NISR), 2007.

[16] Litchfield D. Oracle Forensics: Part 3 Isolating Evidence of Attacks Against the Authentication Mechanism[J]. NGSSoftware Insight Security Research (NISR), 2007.

[17] Litchfield D. Oracle Forensics Part 4: Live Response[J]. NGSSoftware Insight Security Research (NISR), Next Generation Security Software Ltd., Sutton, 2007.

[18] Litchfield D. Oracle Forensics Part 5: Finding Evidence of Data Theft in the Absence of Auditing[J]. NGSSoftware Insight Security Research (NISR), Next Generation Security Software Ltd., Sutton, 2007.

[19] Litchfield D. Oracle Forensics Part 6: Examining Undo Segments, Flashback and the Oracle Recycle Bin[J]. NGSSoftware Insight Security Research (NISR), Next Generation Security Software Ltd., Sutton, 2007.

[20] Fowler K. SQL Server Forensic Analysis[M]. Pearson Education, 2008.

[21] Carlson J L. Redis in Action[M]. Manning Publications Co., 2013.

[22] Macedo T, Oliveira F. Redis Cookbook[M]. O'Reilly Media, Inc., 2011.

[23] Redis RDB Dump File Format https://github.com /sripathikrishnan/redis-rdb-tools/wiki/Redis-RDB-Dump-File-Format

[24] Data types http://redis.io/topics/data-types

**Ming Xu** is a Professor in the college of Computer, Hangzhou Dianzi University, P. R. China. He received the doctor degree in computer science and technology from the Zhejiang University in 2004. His research interests include Digital Forensics, Network Security, Social Network and Data Mining.

**Xiaowei Xu** received the B.S. degree in Computer Science from the Zhejiang University City College in 2011. He is currently a master candidate in computer application technology from the Hangzhou Dianzi University, P. R. China. His research interest includes database forensics.

**Jian Xu** is a Professor in the college of Computer, Hangzhou Dianzi University, P. R. China. He received the doctor degree in computer science and technology from the Zhejiang University in 2004. His research interests include Location-based Services, Mobile Computing, Distributed Database and Network Security.

**Yizhi Ren** is a Lecturer in the college of Computer, Hangzhou Dianzi University, P. R. China. He received the doctor degree in computer science and technology from the Dalian University of Technology in 2011. His research interests include Network Security, Social Computing and Evolutionary game.

**Haiping Zhang** is an Associate Professor in the college of Computer, Hangzhou Dianzi University, P. R. China. He received the master degree in Computer Software and Theory from the Hangzhou Dianzi University in 2005. His research interests include Digital Forensics, Network Security, Social Network and Corporate Information Technology.

**Ning Zheng** is a Professor in the college of Computer, Hangzhou Dianzi University, P. R. China. His research interests include Network Security, CAD, and CAM.