6th International Conference on Cybercrime Forensics Education & Training (CFET 2012), Canterbury Christ Church University, Canterbury, UK, 6-7 September 2012. https://www.canterbury.ac.uk/social-and-applied-sciences/law-criminal-justice-and-computing/docs/cfet/cfet-2012.pdf.

FIREFOX BROWSER FORENSIC ANALYSIS VIA RECOVERY OF SQLITE ARTEFACTS FROM UNALLOCATED SPACE

R. Bagley, R.I. Ferguson and P. Leimich

University of Abertay Dundee, School of Engineering, Computing and Applied Mathematics,
Dundee, DD1 1HG, United Kingdom
<robert.bagley@live.co.uk, ian.ferguson@abertay.ac.uk, p.leimich@abertay.ac.uk >

Keywords: Digital Forensic analysis; recovery of database records, Mozilla Firefox browsing history; SQLite; file-carving; unallocated space

ABSTRACT

A technique and supporting tool for the recovery of browsing activity (both currently stored and deleted) from current and recent versions of the Firefox web-browser is presented. The approach is based upon applying file-carving techniques (matching regular expressions against raw disk images) to the problem of recovering recognizable fragments of internal database structures. The generality of the technique is discussed: It is potentially applicable not only to Firefox but also to any software that uses the popular SQLite embedded database engine such as the Apple Safari web-browser and many Android apps.

1. INTRODUCTION

The reconstruction of web browsing activity is a well-recognised problem in digital forensics. Both commercial and open-source solutions for IE, Firefox and other browsers have been available for some time. Why then, is this a problem worth revisiting? The developers of the Firefox browser have recently moved to a “rapid release” schedule which sees new versions of the software emerge every six weeks; a schedule with which the tool vendors struggle to keep pace. As part of this evolution, Firefox now uses a series of SQLite database tables to store details of browsing history, cookies, favourites etc. Perhaps the most common recovery approach to date has been to export the SQLite files used by Firefox and examine them with open-source SQLite tools. Whilst this technique will extract the current contents of the various database tables in which Firefox records its activity, it makes no attempt to recover deleted records which Firefox zero-fills. Recovery of browsing activity even after its deletion from the database is however possible due to the journaling approach to handling database updates employed by recent versions ($\geq$ 3.7.0) of SQLite: i.e. the use of write-ahead log (WAL) files.

The aim of this work was to recover deleted records from the SQLite database table that held the browsing history by developing a technique and automated program capable of doing this. The table in question is called moz_places and is located in the places.sqlite database file.

The technique presented here involves the analysis of unallocated disk space to recover fragments of the SQLite journaling files and hence the records associated with Firefox activity. The approach, which has been evaluated both as a manual procedure and embedded in a software tool, comprises three stages:

1) Identification Stage: Potential records are located by performing a search for a sequence of bytes that consists of two constant values that appear in all moz_places records, followed by a URL protocol that appears at the beginning of the records' URL field.
2) Verification Stage: Further bytes that exist within the potential moz_places record are examined to filter out any false positives that were identified. Predetermined values and ranges are compared to these bytes to ensure only genuine records are recovered.
3) Reading Stage: Finally, once an authentic moz_places record has been located, its contents are read and extracted to a text file. An additional program was developed to organise the recovery results into a XML table and a suitably formatted summary document.

To evaluate the approach, Mozilla Firefox 10.0.2 was used over a period of two days to browse the Internet and local files. Subsequently the browsing history was deleted. A routine forensic disk imaging procedure was followed and the resulting image examined using our technique. A total of 455 records from an original 469 records were recovered from unallocated space which was a recovery success rate of 97%. Although it was initially designed to recover deleted browsing history from Firefox version 10.0.2, the developed tool has the potential to function on any Firefox version from 4.0 upwards, provided that future versions continue to use SQLite and its journaling files.

The conclusion of this project was that it is possible to recover individual deleted records of the Firefox browsing history in an accurate and forensically sound manner.

2. BACKGROUND

2.1. Browser history reconstruction

The reconstruction of browsing history is a well-recognised technique in digital forensics, a fact attested to by the numerous texts which cover it including Casey (2010) and Jones et al (2006). The motivation for this work was the need to forensically reconstruct browsing history from the Firefox browser. Many tools (both commercial and open-source) are available that will analyze Internet Explorer histories but existing approaches to Firefox are based around either a) using the SQLite database utilities or b) using third-party tools such as SQLite Forensic Reporter (SimpleCarver, 2012) or SQLite Viewer (Oxygen Software Company, 2012). Such approaches concentrate on the files in which SQLite stores its data and hence on the current state of that database, however, work on database forensics including that of Stahlberg et al (2007) and Frühwirt et al. (2010) suggests that by combining knowledge of a database's internal representation schema with the techniques of file carving, it is possible to recover fragments of the previous states of database from unallocated disk space. This raises the possibility of being able to (at least partially) recover deleted items from a Firefox browsing history.

The work of Pereira (2009) and of Riis (2011) concentrates on carving complete SQLite files. Our approach differs in that it applies knowledge of the internal data representation formats used by both the SQLite database and by the host application (in this case Firefox) to guide the carving process at record granularity. This approach is thus also suitable where only fragments of a database are extant in unallocated disk space.

The authors' motivation for this project stems from the ever-growing need for advancements in forensics. Technology is constantly advancing at a phenomenal rate and for a forensic investigator to assist in upholding the law, they must remain up-to-date with all new developments in computing.

2.2. SQLite Databases

SQLite is an open source embedded relational database management system, created by D. Richard Hipp and released to the public in August 2000. It quickly became a very popular leading to many organisations adopting SQLite into their projects, for both commercial and open source products. By these estimates, we see at least 500 million SQLite deployments and about 100 million deployments of other SQL database engines (SQLite, 2012b). These estimates are very rough however it is very likely that SQLite is currently the most commonly deployed SQL database engine on the planet.

The SQLite database system has "a reputation for being highly portable, easy to use, compact, efficient, and reliable" (Allen and Owens 2010). It does this through its "software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine" (SQLite 2012a). This means that unlike other database systems, SQLite does not use a database server to manage the connections to its file. Any process that wants to access a SQLite database simply reads and writes directly from the database files on disk. This benefits SQLite as it requires fewer resources and support from the host computer. The database engine will therefore not have a "separate server process to install, setup, configure, initialize, manage, and troubleshoot" (SQLite 2012c). The simplicity of SQLite allows for the entire database to be contained within a single file. That single file contains the database structure (or metadata) as well as the actual data held in all the different tables and indexes (Kreibich 2010). By default, when database records in SQLite are deleted, the space they had once occupied still contains the records, however the space is no longer recognised by the database and would be considered as free space for new data to overwrite. An exception to this is the moz_places table which has its deleted data explicitly zero-filled, which means that all traces of the data are removed by replacing all of its bytes with zeros. Therefore, records deleted from this table cannot be recovered using conventional methods or the approach of Jeon et al (Jeon 2012).

Newer versions of SQLite use a transaction control mechanism called "write-ahead log" (WAL) to temporarily store its database pages that were altered. An SQLite database consists of one or more pages and these pages can contain records. When a transaction takes place, the database page involved is copied to WAL where the changes to it will take place. This means that the original content of the page initially remains unaltered in the database file and ensures that no errors occur during a transaction. Once the transaction has completed, the database is update by deleting the altered pages it held and replacing them with the altered one from the WAL. The WAL file is then deleted, however as it is not zero-filled, it is possible to the recover its contents from the underlying file system's unallocated space subject to the usual provisos about its overwriting by the normal functioning of the operating system.

2.3. SQLite in Firefox

Version 3.0 of the Firefox web browser, released by Mozilla released in June 2008, was the first to employ SQLite as its underlying database system. Since the introduction of Firefox 3.0, many versions of the web browser have been released, all using SQLite. The current version, at the time of this project, was Firefox 10.0.2 which was released on the 31st of January 2012 and forms the basis for this paper.

Firefox accounts for approximately 25% of all browser installations (Keizer 2012) and is thus regularly involved in digital forensic investigations. From the contents of its databases, a forensic examiner can reconstruct a user's activities on the web and determine if any illegal actions have been performed. However, if the SQLite databases have had their records deleted, this will leave an examiner with only empty databases to inspect.

Firefox uses a number of SQLite databases to store data on the user's activities. This includes information on browsing history, bookmarks, downloads and much more. Out of all the database files, the one considered to be the most important to a forensic examiner is places.sqlite as it holds the moz_places table.

Each record in the moz_places table contains a URL address of a web page or local file accessed through Firefox. It also stores the title and visit count of the URL as well as an ID number for the record. The record contains other fields that could be of forensic interest but it is the former that are of the most importance.

2.4. Forensics of SQLite in Firefox

From a forensic point of view, recovery of the database records is very important whether they were deleted through normal use of the browser or intentionally deleted by the user (Pereira 2009). In either case, it has been established that the content of each record will be zero-filled, making the only possible method of recovery, the deleted data of WAL. When data is deleted it will be moved into the file system's unallocated space, which is an area of storage space that is unused by the computer's file system and free to be overwritten by new data. The process of "data carving" can be used in this situation to retrieve data from unallocated space. Data carving techniques are used during a digital investigation when the unallocated file system space is analyzed to extract files (DFRWS 2006). The structure of a file system is ignored during this process and instead the values of bytes are used to identify data in unallocated space. Therefore the bytes that make up a database record can be used to locate the records in unallocated space if they are known.

2.5. Internal Structure of a Database Record

In order to recover SQLite database records through data carving, it is necessary to understand their internal structure. As shown in Figure 1, the sequential structure of each SQLite record consists of record size, key value, header and data segment (Pereira 2009).

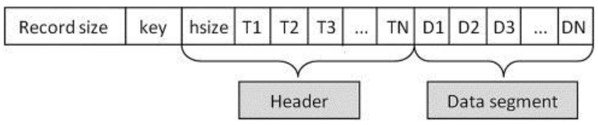

Figure 1. Record Structure (Pereira 2009)

The record size field stores the total size of the record and the key holds the Row ID for the record in the table. The header is comprised of the header size (hsize) which is the size of the header in bytes, followed by the record format (T1-TN). The record format consists of the size in bytes of each of the values in the data segment. Finally, the data segment stores the record's actual field values. In a moz_places record there are nine fields, so there are nine values in the header's record format which can be used to identify how many bytes of the data segment make up each field value.

3. METHODOLOGY

This project sought to develop a technique and computer program to recover deleted records from the moz_places table through unallocated space. However, before the development of the technique could begin, it was first necessary to examine the moz_places table in more detail.

A detailed examination of the moz_places table was carried out in order to provide a basis for the development of the technique. During the examination, it was discovered that two additional fields had been appended to the table: Last_visit_date and GUID. Both these fields had been added to the structure of the table in Firefox version 4.0, and had not been mentioned in previously published papers (which usually considered Firefox V.3).

A WAL file for the places.sqlite database was found in the same directory as the database. Since the moz_places table existed within this database, this proved it was possible that records are present in unallocated space.

The aim in developing a recovery technique was to allow a forensic examiner to manually recover moz_places records from unallocated disk space and to provide evidence that that these records can exist in unallocated space. There are three stages in the technique for recovering records: identification, verification and reading.

3.1. Identification Stage

In this stage, a search is performed to locate potential moz_places records from unallocated space. A sequence of bytes that exists within all records is used as the search token: Two bytes with the hexadecimal values 0x06 and 0x25, are the field size values for the Last_visit_date and GUID fields. These bytes are located at the end of a moz_places record's header and were always present. Their values are invariant making them useful for identifying a record in unallocated space. However, with such a short byte sequence many of the search results could be false positives. To avoid this, the sequence used in the search needed to be longer and therefore more characteristic. Following the GUID field size is the start of the data string which begins with the URL field value. A URL always begins with a protocol, followed by a colon and then two forward slashes (://). This can be used to further identify a genuine moz_places record. Though there are many protocols available for URLs only the four most common were selected to be used: HTTP, HTTPS, FILE, FTP. In Figure 2, the ASCII and hexadecimal values of the four possible byte sequences for identifying a record can be found.

ASCII	**Hexadecimal**
637HTTP://	06 25 68 74 74 70 3A 2F 2F
637HTTPS://	06 25 68 74 74 70 73 3A 2F 2F
637FILE://	06 25 66 69 6C 65 3A 2F 2F
637FTP://	06 25 66 74 70 3A 2F 2F

Figure 2. Byte Sequences

While these byte sequences make it possible to identify only moz_places records that use one of the four selected protocols in their URL value, it does establish a method for identifying records n unallocated space.

3.1.1. Verification Stage

Despite their lengthened specificity, there is no guarantee that the byte sequences used in the identification stage are unique to moz_places records and therefore identified sequences could still be false positives. Therefore verification is necessary in order to ensure that only genuine records are recovered. This can be achieved by reading additional bytes that are not consistent enough to be considered for identification, but appear in all records in some recognisable form. There are five field sizes prior to the Last_visit_date size which either have a constant value or small range of values in all moz_places records. These fields are Frecency, Favicon, Typed, Hidden and Visit_count. They all hold their values in a single byte, therefore a record could be verified by examining the five bytes prior to the identification sequence and ensuring each byte fell within the known value or range for it. If the point of initial identification (Last_visit_date field size) was considered to be at position "0" then the verification process would check the following (Figure 3):

Position	**Field**	**Expected Value or Range (in hexadecimal)**
-1	Frecency	0x01-0x03
-2	Favicon	0x00-0x03
-3	Typed	0x01
-4	Hidden	0x01
-5	Visit_count	0x01-0x03

Figure 3. Verification Sequences

By comparing these bytes to their expected value or range the verification stage assesses a potential record to determine if it is authentic. Whilst it is still possible that the process could verify data that was not part of a moz_places record this is less likely.

3.1.2. Reading Stage

The final stage in the recovery technique is to read the remaining data in the moz_places record in order to extract the information held in it. At this point, a number of the header values are already known through the identification and verification stages. For the record to be read correctly the remaining values in the header needed to be found, along with the Key and record size. This is done by examining the record's data that exists before the five bytes used during verification. The process of doing this is to read the remaining bytes in reverse until the value of the record size is found, signifying the start of the record. The record size, Key and header data are stored as varints which meant they could use multiple bytes to represent their values. A varint would set the most significant bit of a byte to “1” if the following byte was used to form its value. This process continues until a byte's most significant bit was “0”, signifying the end of the varint. However, since the bytes were to be read in reverse, when a byte with a most significant bit of zero was found it marked the start of the next value and the end of the previous value.

Once the start of the record has been reached, the header size and all field sizes are known. The record size and the Key, which is the Row ID of the record, has also been gathered. The final part of the record left to read is the data segment, the size of each of its values being already known. The value of the header's first field size is the number of bytes used in the data segment to form the first field value, the URL. Therefore, the URL can easily be extracted, and this procedure continues until all the field values have been obtained.

3.2. Recovery Program

Two computer programs were created to automate the recovery of deleted moz_places records. The first performs the actual recovery process described above and the second would organises the results into a presentable manner.

3.2.1. Firefox Recovery Program

This program uses the recovery technique described above as its basis and incorporates most of its elements into its design. When this program is run, it obtains from the user the disk image file they wish to recover records from and the location for the output file. Once complete, the program begins searching the source file for any potential record identifiers and if one is found it attempts to verify, read and then output the record to its text file. The process then repeats until the entire disk image file has been examined.

3.2.2. Output Filter Program

This program was designed to take the raw results of the recovery program and give them a more user-friendly appearance. Two files are output by this script and both have a unique purpose. The first is a text document called "resultsout.txt" and contains a defragmented and simplified version of the first program's output file. The second file is an XML file called "resultsxml.xml" which stores the raw results in an organised structure. To allow for the XML file to be viewed in a table format an HTML file was created which when run in a web browser displays the XML file as a table.

4. RESULTS

The execution of the manual technique on the disk image was carried out and a number of records that would have once existed in the moz_places table were uncovered. The total number of records available from the image was unknown, as the technique was designed to be performed manually and it would take an incredibly long period of time to recover all records. The discovery of a single record belonging to the moz_places table was enough to prove that deleted records could be recovered from unallocated space. The technique could also identify, verify and read records from unallocated space which made it possible to use its design to create the recovery program.

4.1.1. Testing the Recovery Technique

The technique was tested by running it against a disk image that had been used to browse the Internet with Firefox before having its browsing history deleted, which would clear the data in the moz_places table. The hard drive used in testing had been zero-filled prior to testing and the same initial installation setup was carried out on each test to ensure all work was performed in a forensically sound manner.

4.1.2. Testing the Recovery Program

The recovery program was tested by executing it on a 30 GB and a 10 GB disk image. The 30 GB image had been used to browse with Firefox for 4 days and contained 857 moz_places records, while the 10 GB image was used for 2 days and had a total of 469 records.

The recovery program managed to recover a total of 11953 records from the 30 GB disk image while 13107 records were recovered from the 10 GB image. This was more records than had originally existed in each image's moz_places table, therefore many of them were found to be duplicates. This is due to multiple deleted WAL files existing in unallocated space.

When the output filter program was run on both sets of results from the recovery program, the number of original records recovered could be obtained. It was found that 285 records out of the original 857 that once belonged to the 30 GB image's moz_places table had been recovered. This meant only 33% of the records had been found in unallocated space. For the 10 GB image, 455 records from the 469 that had existed in the table were recovered. Impressively, 97% of the records had been obtained from this image's unallocated space.

5. DISCUSSION

The recovery technique was successful at proving deleted moz_places record do exist in unallocated space by managing to recover them. The technique was thus automated by the creation of a recovery program.

There was a 64% difference in the number of records recovered by the recovery program for the disk images. Only a third of the records deleted from the 30 GB image were obtained, while nearly all were recovered from the 10 GB image. Both images had been created in a similar manner, the only differences being their size, the days spent browsing with Firefox, and the number of records in their moz_places tables. The 30 GB image was three times the size of the 10 GB image, and as a result had three times the amount of unallocated space. However, this was unlikely to cause any difference in the results, since the records in unallocated space only used a small portion of it. The 10 GB image represented browsing with Firefox for half as many days as the other image. This meant there was only half as much time for records in unallocated space to be overwritten, which could be a reason why the 10 GB image recovered more records. With twice as many records in its moz_places table, the 30 GB image would have to recover 831 records to arrive at the same success percentage as the other image. Since the 30 GB image had more records over a longer period of time, it was likely many of them were being overwritten in unallocated space leading to the poor recovery results. Nevertheless, it is not clear which variable or variables attributed to the difference in the results, but it can be said that the recovery program was not the cause of this issue. This was due to the program being designed to examine every single byte in the disk images and in doing so, no record would be overlooked.

6. CONCLUSION AND RECOMMENDATIONS

The existence of deleted history records in unallocated space has been proved and a technique and tools necessary to recover them have been derived and verified.

The aim of this project was to develop a technique and automated program that was capable of recovering deleted browsing history from Firefox version 10.0.2. This was to be done by inspecting the unallocated space of a disk image for SQLite database records that once belonged to the moz_places table.

The technique was verified by using it to manually recover a number these records thus proving their existence in unallocated space.

With the knowledge gained from the use of the technique, a computer program was developed to carry out the recovery technique automatically. This was achieved, leading to thousands of records to be carved out of two disk images, many of which were duplicates that were then filtered out.

The refined results of the recovery program examination of the two disk images were very different and could only be speculated on. Further work is required to gain a statistical understanding of the persistence of these records in typical operation.

It was found during the examination phase of the project that two additional fields appeared in moz_places in Firefox version 4.0 and above. It is therefore possible that the technique and program could recover deleted records from other versions of Firefox aside from version 10.0.2. Testing both of them on older, and newer, versions of Firefox could also be performed at a later date. At the time of writing, the current version of Firefox, version 14, uses SQLite version 3.7.11. As this continues to use the journaling approach, it is likely that the recovery program continues to be applicable.

SQLite is employed in many applications other than Firefox, notably Apple's Safari Web-browser and Google's Android operating system. The same technique for deriving search expressions should allow the recovery of data from WAL files associated with those applications and may thus open the door to the forensic analysis of a wide range of systems. It is the authors' intention to further investigate this possibility.

References

Allen, G. and Owens, M. (2010), The Definitive Guide to SQLite, Springer-Verlag, New York.

Casey, E.(Ed.) (2010), Handbook of Digital Forensics and Investigation, Elsevier Academic Press, Burlington.

DFRWS (2006), DFRWS 2006 Forensics Challenge Overview, http://www.dfrws.org/2006/challenge/, accessed 25 July 2012.

Frühwirt, P., Huber, M., Mulazzani M. and Weippl E. (2010), InnoDB Database Forensics, 24th IEEE International Conference on Advanced Information Networking and Applications, AINA 2010, Perth, Australia, 20-13 April 2010, Perth, Australia, pp.1028-1036.

Jeon, S., Bang, J., Byun, K. and Lee, S. (2012), A recovery method of deleted record for SQLite database, Personal and Ubiquitous Computing, 16(6), 707-715.

Jones, K.J., Bejtlich R. and Rose, C.W. (2006), Real Digital Forensics: Computer Security and Incident Response, Addison-Wesley, Upper Saddle River, NJ.

Keizer, G. (2012), Silent update speeds Firefox 14 uptake, Computerworld, 24 July 2012, available at http://www.computerworld.com/s/article/9229525/Silent_update_speeds_Firefox_14_uptake.

Kreibich, J.A. (2010), Using SQLite, O'Reilly Media, Sebastopol.

Oxygen Software Company (2012), Oxygen Forensic Suite: SQLite Viewer, http://www.oxygen-forensic.com/en/features/sqliteviewer/, accessed 25 July 2012.

Pereira, M.T. (2009), Forensic analysis of the Firefox 3 Internet history and recovery of deleted SQLite records, Digital Investigation 5(3-4): 93-103.

Riis, R. (2011), Chrome SQLite DB Finder v1.4, online: Guidance Software's enscript download center(restricted access).

SimpleCarver (2012), SQLite Forensic Reporter, http://www.simplecarver.com/exchange/articles/article-41.html, accessed 25 July 2012.

SQLite. (2012a). About SQLite. Available: http://www.sqlite.org/about.html, accessed 25 July 2012.

SQLite. (2012b). Most Widely Deployed SQL Database, http://www.sqlite.org/mostdeployed.html, accessed 25 July 2012.

SQLite. (2012c). SQLite Is Serverless. http://www.sqlite.org/serverless.html, accessed 25 July 2012.

Stahlberg, P. Miklau, G. and Levine, B.N. (2007), Threats to Privacy in the Forensic Analysis of Database Systems, SIGMOD '07, June 11-14, Beijing, China, pp 91-102.