2006

Enhancing the Forensic ICQ Logfile Extraction Tool

Kim Morfitt
Edith Cowan University

Enhancing the Forensic ICQ Logfile Extraction Tool

Kim Morfitt
Edith Cowan University
Western Australia
kmorfitt@student.ecu.edu.au

Abstract

Programmers of forensic tools need to ensure that their tools are of suitable use, robustness and correctness for their output to be used as evidence. One tool for logfile extraction that is currently under development and is intended for forensic use extracts information from ICQ clients has several limitations that need to be overcome before it is of significant value to forensic investigators. This paper covers the process and research involved in further developing the tool, and overcoming a subset of the limitations of the tool. It also documents what was learnt in the process about the logfiles and the extraction tool and provides a snapshot of its current state of development. Also highlighted are the main areas for future development, area's where research is needed, and areas where research could be performed that were highlighted by the current research and development cycle.

Keywords

Forensics, Logfile, ICQ, Instant Messenger

INTRODUCTION

Instant messenger programs allow people from all over the world to communicate in real time over the Internet. The main feature of instant messaging programs is that they allow users to send messages to each other in real time, hence the term instant messenger. There are a range of additional features offered by these programs, including the ability to voice chat, send and receive files, and to use webcams.

These programs may be used to conduct illegal activities over the Internet, allowing people who might conduct illegal activities to communicate more easily, such as hackers and other criminals. Pedophiles also use the Internet to search for victims. There are even programs available to allow parents to control their children's activities online, such as NetNanny (NetNanny).

ICQ is one such instant messenger program, and has all of the above features. It is also capable of logging the conversations of users, and other activities they perform online using ICQ. Logfiles from chat programs can provide valuable information on criminal activities to forensic investigators.

The format of the ICQ logfiles is in binary, and the structure is not widely known, making analysis of the files for forensic purposes quite difficult and time consuming to perform manually. There are a number of different versions of the official ICQ client and several different versions of the logfiles, which further complicates the issue of logfile analysis.

It is for this reason that in 2004 a project was undertaken to find or create a tool that could automatically extract the entries in the logfiles into a format more useful to forensic investigators. This project successfully created a program that would extract parts of the logfiles. There were, however several limitations in the program which limit its usefulness.

This paper describes the limitations of the existing program and the process undertaken to extend the program to a level where it would be more useful to forensic investigators.

THE EXISTING PROGRAM

After initial research the project found that there was very little in the way of existing programs that could be used to fulfill the requirements set out for completion of the project. A program called icqhr (HitU, 2003) was found which could be used to compare the output and assist in verifying the accuracy of the tool, however it is a closed source program created by a Russian hacking group, which limits its forensic credibility. Also it did not extract partial records, a requirement of the system.

Research also uncovered work performed by Derek Soeder(2000) and Strickz(2002), which provided most of the basics of the structure of the logfiles and allowed work to begin on understanding the logfile structure. From this work algorithms were created, tested, and modified until the raw records could be extracted. Further information was also discovered about the structure of the logfiles, such as how to reconstruct them from partial records and how to identify the version of ICQ that created a logfile from the record header of a single logfile record.

The results of the project were presented at the end of 2004 and the project was deemed to be successful. The structure of the logfiles was now known, differences between different versions of the logfiles were known, and a program had been written that could output one of the main components of the logfiles, the messages sent between users.

LIMITATIONS OF THE EXISTING PROGRAM

There were several limitations to the program that was written that limited it's usefulness. The latest version of ICQ 2003b, is quite different in structure to the previous versions of ICQ, and so the program could not extract records from logfiles created by that version of ICQ. Of the several different types of records that can exist in a logfile, only the records relating to messages can be output. While messages are extremely important, there other logfile entries, such as user information, and contacts that could be extracted.

Although all records are extracted from the logfile, only certain messages are output. When a record is deleted from the logfile, it is not actually overwritten but simply removed from the index file that is associated with each logfile. The record remains in the logfile until its space in the logfile is required by another record. At that time a deleted record may be partially or wholly overwritten. If two records overlapped there was no method of validating records or knowing which record was written first without using the index file. Records that may have been partially overwritten are not output.

The records that are output are not sorted in any way. It would be up to the user to sort the records manually, a task which would be time consuming and much better performed by the tool.

Finally, although the extraction algorithm works, there are assumptions that have been made that may not be correct in all cases, even though they were adequate for the test data that was available. The first of these is that the first record encountered in the logfile is always valid. This assumption was arbitrary, and made to keep the project on schedule.

The second assumption was a coding artifact and was not immediately obvious. When the start of a record could not be verified, the length of the record stated in the record could not be relied upon. This is due to the fact that the first part of the record is the length of that record, and any overlap overwrites at least part of the length. Therefore there was no way of knowing where the record finished. To ensure all of the record was obtained, everything was taken up to the start of the next record as part of the current record, or if there were no more records, everything was taken the end of the file. Records were identified by a signature found in every record. This signature starts 13 bytes into the record. If a signature is found then it is known that for that record, that the signature is reliable, however no effort was made to ascertain the validity of the previous 12 bytes, it was assumed that they were good.

This does not take into account a situation that may exist where one record is deleted, the first few bytes are overwritten with a new record, and that record is then deleted and partially overwritten with a third record. The newest record would be found, and the second record found, and it would be shown that they overlapped. The length of the second record could not be trusted, a search would be made for another signature, which would find the first record written. It is here the assumption would incorrect. The start of the third record we know to have been overwritten, however the program assumes it to be correct, so subtracts 13 bytes from the position of the of the third records signature and ends the second record there. It then reads the length of the third record, assuming it is valid.

From that we can see two things. Firstly part of the second record has been missed, and length of the third record is incorrect. While no errors were detected as being caused by this problem in testing, it is clearly incorrect and a new method of determining which record is the one overwritten needed to be created. Also it meant that if the program was enhanced to dump raw records, manual analysis would be required to determine which record is the one overwritten.

THE ENHANCEMENTS

From this list of limitations, three were selected to be corrected. They were chosen due to their ability to correct potential errors, enhance the correctness of the program and to enhance it's usefulness to forensic investigators.

The limitations selected were:

1. Lack of sorting of records.
2. Validity of the first found record to verify correctness of extraction.
3. Validation of extracted records to improve extraction and verify correctness of extraction.

By concentrating on verifying the extraction algorithm, the existing program can be used with more confidence that the work that it does perform is correct, and that future enhancements build on this.

Sorting of the records simply makes the tool easier to use by forensic investigators and saves time when testing results against icqhr as both sets of output are then sorted in a similar fashion.

SORTING RECORDS

All that needed to be decided upon before implementing sorting was the sort order. The sort order decided upon was by the UIN number which identifies other ICQ users, and then by time, giving for each user that the person under investigation has communicated with using ICQ, a chronological output of their interactions.

From there the process was a matter of implementation. The extraction tool is written in C++, a language which has easily extendable sorting features. A unit of code called a class was written which decided the precedence of two records. From there it was plugged into the existing program code. Testing showed the sorted output to be blocks of messages, each one a chronological output of all records found for a particular UIN found in the logfile.

VALIDITY OF THE FIRST FOUND RECORD

Another potential problem that reduced the certainty of the program was the fact that it was not known if the first record found would always be valid. The existing information gave no indication as to the position and type of the first record. There was information suggesting that the logfile had a header and was organised by pages in a similar fashion to the index file. This suggested that the first valid record may occur at a file offset of 0xCD, as this was the first byte after the first page header. When this position was checked in the logfiles used for testing, no record was found there.

Several logfiles were searched. Some were logfiles that were completely new and had little or no user information in them other than a My Details record. One was a logfile generated for testing purposes, and had seen extremely limited use. The last was the logfile of the author and has been used for an extended period of time, and has many records deleted from it. In all cases, the first record was found to be at the file offset of 0x555 and had as the first byte of its signature 0xE9. This is significant as this record has been identified as being used for sound settings. Even in the logfiles that had little or no changes to the sound settings this record existed as the first record in the logfile.

In doing previous research, logfiles were generated from different versions of the ICQ client, with just the initial entries that are created by default when the user first connects using that client. A number of these were checked all were found to have this record as their first entry, and were at offset 0x555. In its initial state, the record simply had its record header, and was followed by 8 bytes of 0x00. In the author's logfile, the record was much larger, appearing to contain a lot of information about wave files and sounds to play when different events occur, such as the user receives a message.

Another point of interest is that this record has a particular entry number allocated to it, 1007 or 0x3EF in hexadecimal. There are several types of records that are unique in a logfile, such as the Address Book, which has an entry number of 1006, or 0x3EE in hexadecimal, and the My Details record, which has an entry number of 1005, or 0x3ED in hexadecimal.

Although the sample of logfiles checked was quite small, it was clear from the positioning of the record in the logfiles, and that it has its own entry number, that it is highly likely that all logfiles for the ICQ version that this program works on contain this record as the first record in the file.

Therefore as the first record in the file is always a valid record, the assumption that it would be is correct and no longer of issue. There would not need to be any changes to the program, however a check to ensure that the record expected to the be first one found is actually the first one found, and that it is in the expected position could possibly be useful for showing that the logfile was found to be sound.

IMPROVING RECORD IDENTIFICATION USING RECORD NUMBERS

When the original program was designed, it was hoped that adequate extraction could be achieved without having to inspect any other part of the record, other than its length. When it was found that records were simply deleted from the index file and not from the logfile, a method was designed to try to ensure that the records were as complete as possible without having to inspect the records.

This method worked well when there were no partial records, however when two records overlapped there was not enough information available to adequately determine which of the two records overlapped the other. It would be known that two signatures were found, indicating that there were two records, and that the start of the second record, which could be calculated, overlapped with the end of the first.

It is known that all records share the same record header format, and that records for events that represent interaction between different users, such as messages, are time stamped. The format of the header includes an identification number for the record. As mentioned before, some records which occur only once and server a particular function such as the My Details record, have a particular identification number, also known as an entry number, associated with them. Other record types, such as messages, have an identification number assigned when the record is created. If the numbers could be shown to be assigned sequentially, then they could be used to determine which record was written first.

Record time stamps could have been used, however if the system clock is inaccurate, if the user changes the system time, or the system battery is removed, then the timestamps can no longer be relied upon to determine which record was written first.

The record identification numbers have the limitation that if more than 8 bytes of the two records overlap, they cannot be relied upon either. The difference between using the timestamp and the record numbers, is that the program can determine how much overlap occurs. It cannot determine if the system time can be relied upon. This is something the forensic analyst would be required to determine before timestamps can be used to determine which records were written first.

Before record numbers could be used, it had to be determined that they were indeed assigned sequentially. For this a reliable way of ordering the records was needed. In this instance time stamps could be used as the authors logs have been created on a computer that has a utility called StatBar. One function of StatBar is to keep the system time in synch by the use of the Network Time Protocol (NTP). For almost the full life of this logfile, the system time can be relied upon to be correct.

A copy of the existing logfile extractor was created and a class called AscendingSort was created to sort all the message records by their record number. The program was then modified to extract all the records, and create a list with a copy of all the message records found. Only message records were used as they are the only records that were known to exist in this file that required a time stamp. Next the list of messages was sorted using AscendingSort. Lastly the program iterated through the list of messages firstly checking to see that the time stamp of the current record was greater than the time stamp of the previous record. The program also printed out each records record number followed by its time stamp so that its accuracy can be visually checked. If the program found a record that had a time stamp that was lower than the previous record a message was printed to screen.

When this modified program was run on the testing logfiles, it failed to find any timestamp that was out of order, showing that records without a dedicated record number are assigned their record numbers sequentially. Having verified this, the next stage was to create an algorithm to use the record numbers to improve extraction.

AN IMPROVED EXTRACTION ALGORITHM & VALIDATION OF EXTRACTED RECORDS

With the ability to determine which records were created first, some of the problems relating to improving the extraction algorithm had been removed. There remained however the problem of how to verify records where the overlap between records was greater than 8 bytes. More than 8 bytes of overlap makes the use of record numbers to identify which record was written first impossible. The index file could be used if it was available, however there still needed to be a method to determine the precedence of records where the index file was unavailable. The index could still be a useful reference for a forensic analyst if available, as it could provide corroborating evidence as to the correctness of the programs output.

For reasons stated above time stamps were also discounted. They cannot be relied upon to be correct. This left determining if the whole of the record was complete as the only method of determining if which was the first record written. For each different record type, and each different version of a record type, a verification algorithm would need to be created. For existing version of the program, only two different record types, Long Message and Short Message are extracted at the current time. This meant that extraction methods needed to be created for only these types of records.

Due to the nature of the first 12 bytes of a record, the only part of those 12 bytes that can be verified by looking at the rest of the record is the record length in the first 12 bytes. If the end of the record can be verified, then the length of the record can be verified, and subsequently the expected value in the record length is known. If the correct value is found in the record length, the remaining 8 bytes can be safely assumed to be correct. For the purpose of extraction however, the end of the current record is being compared with the start of the next.

To determine if the start of the next record has been overwritten by the current record, it only need to be determined if the end of the current record is intact. From this it can be determined if the start of the next record has been overwritten by the current record. If it has not been overwritten by the current record is likely (although not guaranteed) to be intact.

Using what is known, the algorithm on the following page, displayed in Figure 1 was created. First the file is searched until a signature is found, then a record is created for that signature. Then the next signature is found. A reference is kept to the previously created record. If there is no previous record (first iteration of the loop) the algorithm simply marks the start and end of the record as CLEAN, reads the length of the record from file and reads in the data for the record to store.

If there is a previous record, and the end of the previous record overlaps with the start of the current record, the algorithm checks to see if the overlap between records is equal to or less than 8 bytes. If the overlap is less than or equal to 8 bytes, the record numbers of the two record are compared. The record with the lower record number is considered to be the record written first. This record has its start valid (current record) or end valid (previous record) property set to OFFSET, and the amount that it overlaps (the offset) is stored in the record. The other record has its (current record) or end valid (previous record) property set to CLEAN. As this record is considered to be intact it has no OFFSET. If the overlap is greater than 8 bytes, then the end of both records is set to DIRTY.

If the start of the record is marked as clean, the length of the data is read from the file and four is added to it. This allows for the four bytes that record the length of the record, as they are not already included into the length of the record.

If the start of the record is not marked clean, one of two things happen. If there is no further signature's in the file, the data length is set so that everything is taken to the end of the file. If there is another signature, a method named verifyRecord() is called. This method uses the signature to identify type of record, and then calls the implementation of verification algorithms for those record types that verification algorithms were created for. If there is no verification algorithm the type of record is output to console.

The verification algorithms are used to determine if the end of the record has been overwritten or not. If the algorithm is able to verify that the end of the record has not been overwritten it is marked as clean, otherwise if it is able to verify the portion of the record that has been overwritten and so marks the end of the record as OFFSET and sets the offset value to the amount of the record that has been overwritten.

If the algorithm is unable to verify enough of the record, then it is unable to determine the end of the record. It simply marks the end of the record as DIRTY.

When the verifyRecord() method returns, if the record is marked as DIRTY then the data length is set to take everything up to before the signature of the next records starts. This does two things. First it ensures that all the data possible for that record has been put into that record, and it ensures that the next record is verified, as on the next iteration an overlap of greater than 8 will e detected. This verification is required as there is no way to know is the current record has overwritten any part of the next record, so the record length of the next record cannot be trusted until.

Figure 2 and figure 3 are the algorithms for verification of short messages and long messages. These algorithms determine how much of the record is valid. Each record contains a second separator value and are identical up to the second separator value. Between the signature and the second separator value is the ANSI text of the message. The first check that both algorithms do is to check the second signature matches the first.

Next the short message algorithm validates the end of its record, which is simply 27 bytes of 0x00. If 27 bytes of 0x00 are found then the end of the record is marked as clean. If not, the end of the record is marked as OFFSET, and the first byte not found to be 0x00 to be first byte of the end offset of the record.

The long message verification algorithm varies in that the number of 0x00 bytes at the end of the record is different. The calculation of the position of those bytes is more complicated due to copies of the message that may be stored in rich text format, and/or in UTF8 format between the time stamp and the 0x00 bytes at the end of the record.

Other than these differences, the verification algorithms are extremely similar.

RESULTS OF THE ENHANCEMENTS

The sorted records will simplify and speed up analysis of the output, saving time and reducing the possibilities of mistakes by the forensic analyst.

Knowing that the first record in a logfile is almost certainly correct verifies what the implementation of the program has assumed, improving the validity of the information and its forensic usefulness.

On the test data that was used, which included a file with over 2000 records, the algorithms verified all but two of the records. The research resulted in a stronger and more verifiable algorithm with more accurate information about the validity of the data stored in the records. Further testing of the implementation of the improved algorithm is required, however, and further checks need to be added to ensure that all of the extracted data is correct. Some of these tests can be tests that are compiled out in production code, by using a feature of C++ called an assertion. An assertion is a simple boolean test that will throw an exception and halt execution of the program if the test returns false.

Execution of the program and debugging has shown more records that exist in the test logfiles that were not previously identified, as they were simply ignored in favour of the message records.

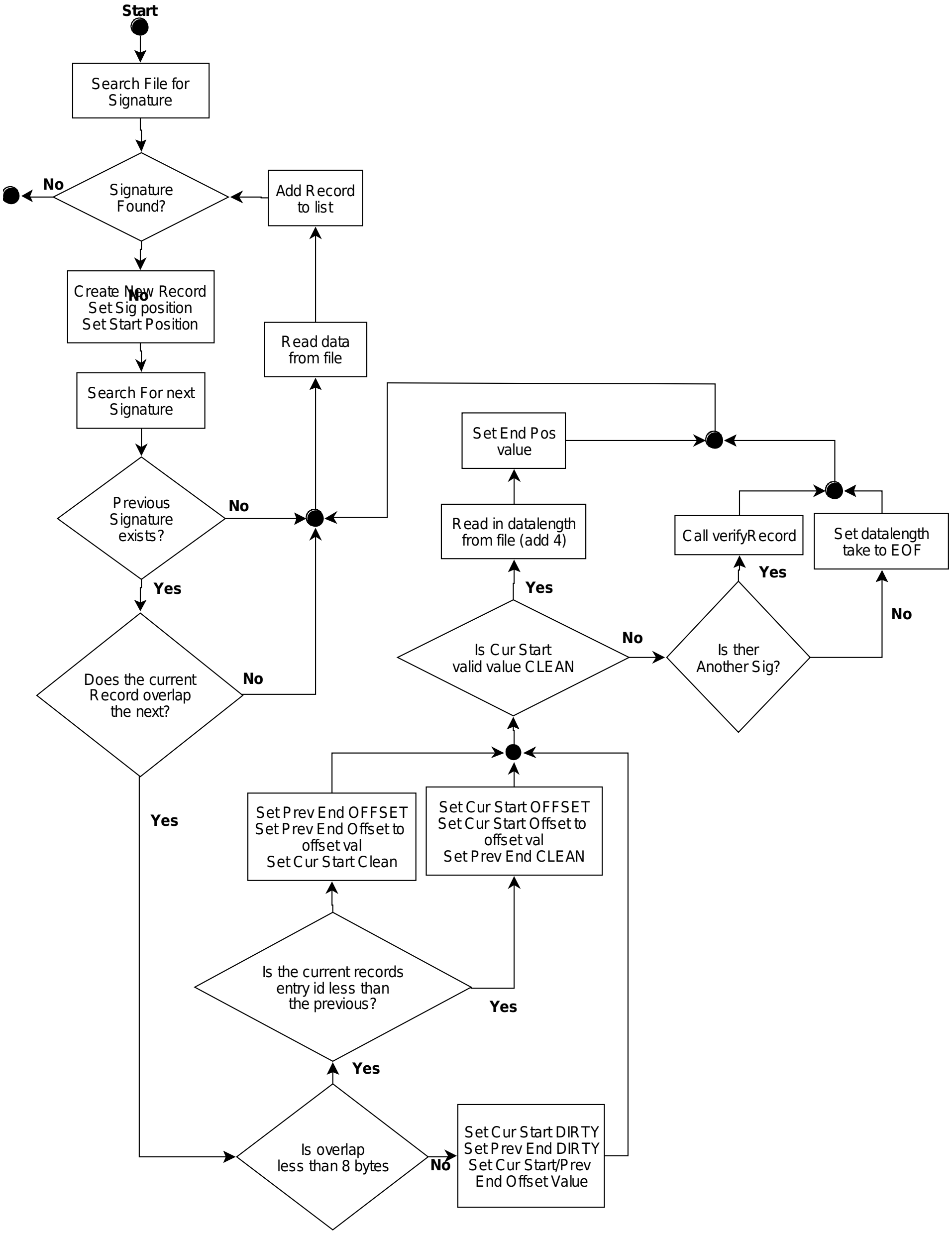
Start
Search File for Signature
Signature Found?
No
Add Record to list
Create New Record
Set Sig position
Set Start Position
No
Read data from file
Search For next Signature
Set End Pos value
Previous Signature exists?
No
Read in datalength from file (add 4)
Call verifyRecord
Set datalength take to EOF
Yes
Yes
Yes
No
Does the current Record overlap the next?
No
Is Cur Start valid value CLEAN
No
Is ther Another Sig?
Yes
Set Prev End OFFSET
Set Prev End Offset to offset val
Set Cur Start Clean
Set Cur Start OFFSET
Set Cur Start Offset to offset val
Set Prev End CLEAN
Is the current records entry id less than the previous?
Yes
Yes
Is overlap less than 8 bytes
No
Set Cur Start DIRTY
Set Prev End DIRTY
Set Cur Start/Prev End Offset Value

Figure 1: Record extraction algorithm

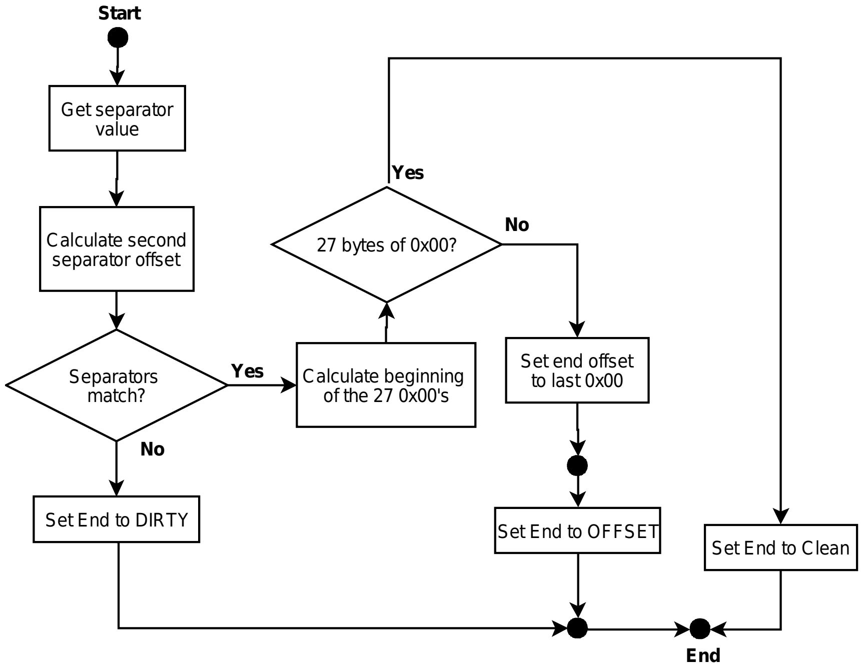

Figure 2. - Short Message Verification Algorithm

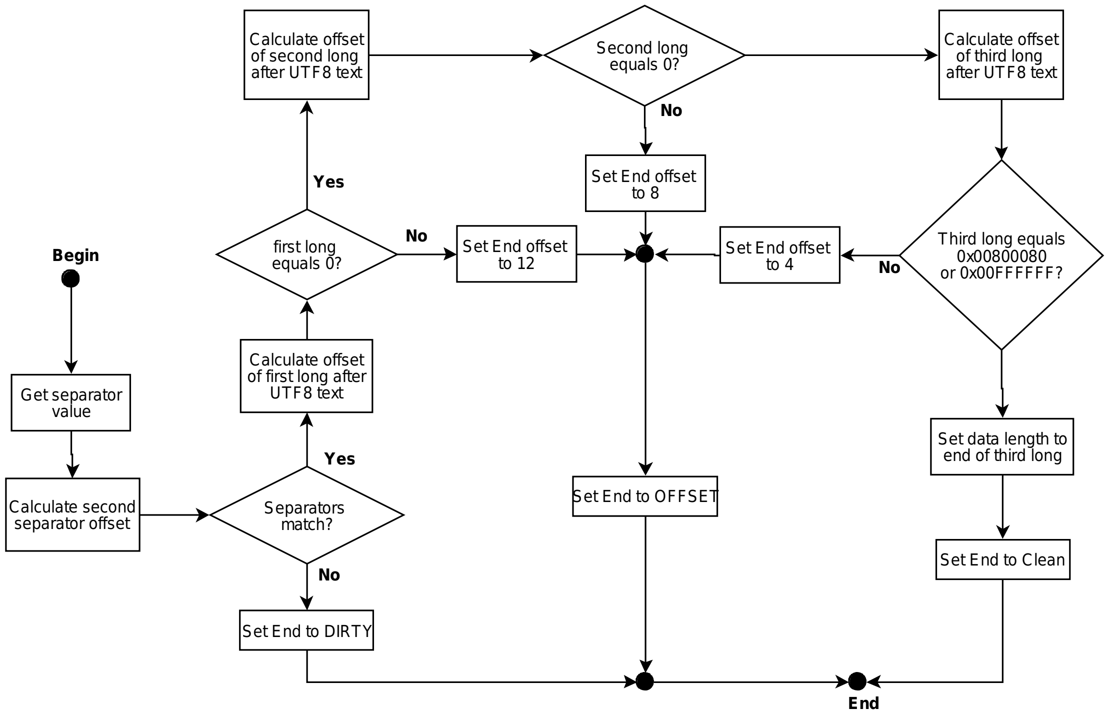
Begin
Get separator value
Calculate second separator offset
Separators match?
Yes
No
Set End to DIRTY
Calculate offset of first long after UTF8 text
first long equals 0?
Yes
No
Set End offset to 12
Calculate offset of second long after UTF8 text
Second long equals 0?
No
Set End offset to 8
Calculate offset of third long after UTF8 text
Third long equals 0x00800080 or 0x00FFFFFF?
No
Set End offset to 4
Set End to OFFSET
Set data length to end of third long
Set End to Clean
End

Figure 3. - Long Message Verification Algorithm

LIMITATIONS OF THE NEW VERSION

There are still several limitations existing with the new version and more testing is required. During testing some areas of the program were noted as needing addressing, but as of yet have not been addressed.

The first of these is that the short message verification algorithm should set the end of the record to DIRTY if none of the 27 bytes can be validated. Currently in the code if none of the 27 bytes are validated, then the end of the record is marked as OFFSET and the offset of the end of the record is set to 27. This would be correct if the previous element in the record was the second signature, which had just been verified, however the time stamp is between the second signature and the first of the 27 byte of 0x00. Setting the offset to 27 bytes assumes that the time stamp is correct which has not and can not be verified.

What this suggests is that there needs to be a record of what part of the record has been verified as intact, as well as what part has been verified as overwritten. The offset value records what part of the record has been overwritten, and with the addition of an element to record what part of the signature has been verified, there would be a section of the record marked as good, a section marked as bad and any section remaining would be regarded as unknown.

The same problem exists for the long message verification algorithm. During documentation and testing it was shown that the checks for the last 12 bytes of 0x00 in the record produce the same problem. It was also noticed that an additional check could be added that could check part of the record after the time stamp, to see if the time stamp can be validated.

The extraction algorithm itself has yet to be subjected to rigorous testing to eliminate any yet to be discovered bugs, and to ensure that all paths through the algorithm produce a record of consistent state, with accurate information. While this has received testing to ensure that the program runs as expected, and that the results that are checked are valid, further validation is required for forensic purpose.

The remaining limitations also exist. No other records have been output and the output of the message records has yet to be altered to output partial records. No work has been performed to extract records from version 2003b of the ICQ client and no work has been done to allow the analyst to account for time stamps.

FUTURE WORK

Firstly more work is needed to test and refine the algorithm. While no major are changes required to complete the current work, extra checks can be included such as the asserts mentioned earlier, as well as checks to ensure that the first record in the file of the type expected, and at the expected position. If the type of the record found is not what is expected, this can be flagged for the analyst to investigate. Once fully tested this algorithm should need no further changes to extract records by signature.

Further work can possibly be done to extract text strings from the file. This may be required to be a separate extraction pass that requires parameters from the user, and possibly may be interactive or partially interactive. This is due to the nature of how the strings are stored, and the ability to generate a large amount of false positives, which would need the analyst to adjust parameters to filter out.

Also the remaining record types need to be output, as do records from 2003b. For versions of the ICQ logfiles created from 2003b, this will require a great deal of research to ascertain the structure of the logfiles.

For versions of ICQ earlier than 2003b, this will be a matter of implementing the record types and adding them to the program.

Finally, while writing this paper, a possible technique for analysing the history of the logfiles in regards to deleted records and when they were deleted is in the early stages of being concocted. This technique would use the positioning of records of different numbers in the file, and what records are overwritten by a series of records to make determinations on when they were deleted. This information may be of limited use, however it is possible that it may give information about actions by the user such as when the user deleted a contact and history from the logfile.

A technique such as this would require knowing exactly how a section of a file that contains a deleted record is selected to be overwritten and if time stamps cannot be reliably used, may only give information relative to other records. This information, if it can be determined might, however provide supporting evidence as to the actions of a subject under investigation. Even if it is found that there is no possibility of reliably using this information, knowing when a record is overwritten and the criteria for the reuse of this space would be useful as background information when using logfiles as evidence, and may provide other useful information.

CONCLUSION

While the current additions to the logfile extractor are only in the early phases of testing, the program is showing itself to be ready for more rigorous testing. There is still a lot of work that needs doing to build up the program into a more useful tool, however as a preliminary investigation tool, it could be quite useful.

Further research into the latest versions of the official ICQ client are required to document the structure of the logfiles and to allow the program to be extended to extract records from those logfiles.

Ideas for techniques to analyse numbered and time stamped data are being formulated as topics for further research. While this is secondary to the purpose of generating logfiles that are useful forensically, it can be considered that as the reason the logfile extractor has been created is to improve the forensic analysis of the logfiles, this type of research could be relevant the project and could be undertaken to improve the body of knowledge that should be associated with a program such as this.

REFERENCES

HitU. (2003). icqhr (Version 1.8f).

NetNanny. Retrieved Novemer 3rd, 2005, 2004, from http://www.netnanny.com/p/page?sb=product

Soeder, D. (2000). ICQNEWDB. Retrieved Jan 20th, 2004

Strickz. (2002). ICQ Db Specs. Retrieved Jan 21st, 2004, from http://cvs.sourceforge.net/viewcvs.py/*checkout*/mirandaicq/Plugins/import/docs/import-ICQ_Db_Specs.txt?content-type=text%2Fplain&rev=1.9

COPYRIGHT