A Forensic Log File Extraction Tool for ICQ Instant Messaging Clients

Kim Morfitt
Edith Cowan University, Western Australia

Craig Valli
Edith Cowan University, Western Australia

A Forensic Log File Extraction Tool for ICQ Instant Messaging Clients

Kim Morfitt
Edith Cowan University
Western Australia

Craig Valli
Edith Cowan University
Western Australia

Abstract

Instant messenger programs such as ICQ are often used by hackers and criminals for illicit purposes and consequently the log files from such programs are of interest in a forensic investigation. This paper outlines research that has resulted in the development of a tool for the extraction of ICQ log file entries. Detailed reconstruction of data from log files was achieved with a number of different ICQ software. There are several limitations with the current design including timestamp information not adjusted for the time zone, data could be altered, and conversations must be manually reconstructed. Future research will aim to address these and other limitations as pointed out in this paper.

Keywords: ICQ, instant messaging, logfile, forensic, extraction

1. INTRODUCTION

1.1 Background

ICQ is an instant messaging system by which users can send messages and files to other users, as well as interact in various other ways such as file exchange voice chat and the use of web cams. The official ICQ client logs all conversations and other interactions between users to file in binary format and stores this on the disk of the client machine. There are several different versions of the official ICQ client, with subtle differences between different log files for different versions of the software. This variety of logfiles and software versions and binary nature of the log file makes for difficult and time consuming forensic analysis of these files.

Hackers and criminals often use instant messenger programs such as ICQ for illicit purposes and consequently the log files from such programs are of forensic interest (Mosnews, 2005; Poulsen, 2005). If ICQ itself is not used directly in committing an offence its log files may contain valuable corroborative evidence about actions or activities of the users (Anonymous, 2002).

Instant messenger programs such as ICQ are often used by hackers and criminals for illicit purposes and consequently the log files from such programs are of forensic interest. If ICQ itself is not used directly in committing an offence its log files may contain valuable corroborative evidence.

Prior to this project some work had gone into working out the format of the database files by Soeder (2000), and further work by Strickz (2002). Their work documents the structures of the files and most of the components of the data structures used in the binary log files. The information provided by the two authors that provides good insight into the structure and purpose of the two different log file types identified by their work. While these files impart information about the structure of the log files, there is no information contained within on how to extract the information from the log files. The information in the documents contains information on the data structures in the database.

A program called icqhr (Hitu, 2003) outputs log file information to a html text file, rather than XML and was used to assist in comparing the accuracy of the system under development. The tool has been black box tested and it accurately outputs the data that should be stored in the log file. Although, a Russian "hacking group" has created it, icqhr was invaluable as a reference. This paper outlines research that has resulted in the development of a tool for the extraction of ICQ log file entries.

2. ICQ LOG FILE FORMAT BASICS

2.1 The ICQ Log Files and What They Do

There are two types of files that make up an ICQ users log files. The first is an index file we have called the IDX file. An IDX file is so named as it is the file extension for this file type. The file name itself consists of the users UIN (Users Identification Number) used to access the ICQ servers, followed by ".idx" extension. This index file contains a doubly linked list which serves as an index to the second file.

The second file is the data file which contains the log records. The file name consists of the users UIN (Users Identification Number) used to access the ICQ servers, followed by ".dat" extension. We have called this file the DAT file.

The DAT file is file of forensic interest. The IDX contains no valuable information on its own, however as shown further down, it can be used to reconstruct a DAT file from file fragments, or show which records are valid within a DAT file. The DAT file contains all of the log file information that can be stored by ICQ about a user's interaction with other users on ICQ.

Analysis of the IDX file was easily achieved the file layout was simple, with only a few data structures that required some interpretation. An IDX file was opened in a hexadecimal file editor and the information in the files were

compared to the documentation that was downloaded. This illuminated the fact that the files were written in little Endian format (least to most significant byte). Manual analysis of an IDX file is a time consuming process, even for a small number of entries, and there is a limited amount of information that can be gained from the file. The garnered information however, could be used for cross-referencing and verification of DAT file entries.

To verify the information gained from the analysis was in fact correct, a test program called "IDX Reader" was created to extract information from the IDX files. The purpose of this program was to use the information gathered to extract all the entries from an IDX file and store the information in a linked list. From there the information could be output to screen showing the stored information that was extracted.

2.2 Possible Reconstructing of Log Files from Fragments

This paper presents ideas regarding possible methods for reconstructing ICQ log files. First a caveat, due to the fact that not everything is known about the log files, and the limited testing data, some of the information presented here may not work in all conditions or versions of ICQ, even though the IDX file structures appear to be the same over all versions of ICQ that implement this method of logging. The files used when forming these techniques had little or no previous data deleted from them i.e they may not be representative of extended and extensive use by user. It may be that deleted data affects the structure of the IDX files, and the ordering of these records. This would affect attempts to reconstruct IDX files. This possibility is currently a matter for further research.

IDX files contain a doubly linked list of offsets to valid entries in the corresponding DAT file. These entries are numbered. Once the corresponding entry is located in the DAT file and identified by its number, its position within the DAT file is known from the offset given in the IDX file. There are two possible modes of reconstruction for the IDX files. One is to attempt to reconstruct the file, and the other is to extract what data can be extracted from the available fragments. The main reason for reconstructing an IDX file is to allow reconstruction of DAT files for forensically sound analysis.

2.3 Reconstruction of an IDX File

Reconstruction of an IDX file is relatively straight forward task if all the fragments of the file are available, which for explanation of the process is assumed. The IDX file itself contains a file header, page headers and the entries themselves.

The file header is 20 bytes in size. It has the format shown in figure 1. As shown, it has a signature of 12 bytes at the start of the header that can be searched for. That signature will be at the start of a file fragment, as it is also the start of an IDX file. A page header will immediately follow the file header.

```
==================================================
== Format of IDX main header (20 BYTES):
==================================================
00000000   3 LONGS    Unknown, but always 4, 20, 8.
(04,00,00,00,14h,00,00,00,08,00,00,00)
0000000C   LONG       IDX pointer to root entry
00000010   LONG       ICQ database version
                 10    = ICQ 99a
                 14    = ICQ 99b
                 17    = ICQ 2000a
                 18    = ICQ 2000b
                 19    = ICQ 2001a, 2001b, 2002a, 2003a
```

Figure 1: Format of IDX file header (Strickz, 2002)

The page headers are 205 bytes in size and are always separated by space for 1000 linked list entries (20Kb). The format for a page header is shown in figure 2. Again the entry begins with a series of bytes, 20 bytes in all, that can be used as a signature to locate page headers.

```
================================================================
== Format of IDX page header (205 BYTES):
================================================================
00000000       5 LONGS        Unknown, but always 201, 0, 0, 0, 0.
00000014       LONG           Pointer to next page header. -1 if this is the last
page.
00000018       LONG           Unknown, always 1?
0000001C       LONG           Number of bytes in each slot (20)
00000020       LONG           Number of fragments in the page with one or more
                              consecutive free slots.
00000024       LONG           Number of empty slots in this page.
00000028       10 LONGS       Unknown, always 0?
00000050       125 BYTES      Allocation bitmap
```

Figure 2: Format of IDX page header (Strickz, 2002)

Initially, the IDX file is created with a file header at offset 0h, followed by a page header at offset 14h. This is followed by space for 1000 20 byte entries, making the initial file size 20225 bytes in size. Additional space for entries is allocated as required. Allocation is done by appending an additional page header, plus space for 1000 entries. From this we know that:

- The size of an IDX file will be a multiple of 20205 bytes plus 20 bytes.

- Beginning at offset 20h, every 20205 bytes there will be a page header

If all the page headers in the fragments are located, then the file size can be calculated. This is done by multiplying the number of page headers by 20205 and adding 20, to get the size of the file. This will only work if all page headers are properly identified.

Alternatively another method is the highest value next pointer can be located and 20205 bytes can be added to that to get the expected file size. The page headers can be ordered in ascending order by their next pointers. The first page header is always located at offset 14h, and the remainder located at intervals of 20205 bytes. To locate position of a page header subtract 4EEDh from the next pointer to find offset at which the header begins. All the page headers can be positioned in this manner. It is possible to locate the position of the remaining fragments by using the positioned fragments containing the page headers. Locate a valid linked list record near the end of a previously positioned fragment. The format of the IDX linked list entry is shown in Figure 3.

```
============================================================
== Format of IDX linked list entry (20 BYTES each):
============================================================
00000000  LONG      entry status? : -2       = valid IDX entry
00000004  LONG      DAT entry number:
00000008  LONG      IDX pointer to next entry (-1 = none)
0000000C  LONG       IDX pointer to previous entry (-1 = none)
00000010  LONG      DAT pointer to corresponding DAT entry (-1 = none)
```

Figure 3: Format of IDX Linked List Entry (Strickz, 2002)

It should be simple enough to order the remaining fragments by the DAT entry numbers in the linked list records contained in the fragments. The higher entry number fragments tend towards the end of the file and it should be possible to reconstruct the remaining fragments into the file in that manner.

If it is possible to search for information in fragments at certain offsets, there is a second method that could possibly be able to identify the correct fragments and be used to automate the reconstruction. This method would use the last linked list record in a fragment that has already been positioned in the reconstructed file, and noting it's next pointer. This pointer would likely point to a linked list entry in the next fragment. The offset into the fragment could be calculated. The signature of a linked list entry is known. It is FE FF FF FF (-2) in the first 4 bytes of the linked list entry.

Use the signature to search for an entry at that position in all of the fragments. If an entry is found there, and it is the next fragment, it's pointer to the previous linked list entry would contain the offset of the last entry in the previous fragment (being the linked list entry used to find the next file fragment).

2.4 Extracting IDX Information Without Reconstruction

The required data can be extracted without reconstructing the file, however it may required to perform additional checks to identify valid records. The only identifier of valid records is the signature FE FF FF FF (-2), which identifies a valid record. Assuming that the signature is not found in any other position in the file (it may possibly be found in the page header bit maps), it should be

possible to dump all the valid records out to another file.

As a result of analysis, three data structures were found in the IDX file namely file header, page header and the actual data. The file itself was found to be highly structured with predictable characteristics. Most of this information was already documented in the downloaded information, however there were a few observations made that, could not be investigated and explained in the time available. The first data structure found was a file header. The file header contains some unknown information, a pointer to the first record data structure, and the version of ICQ database. The file header is always found at the start of an IDX file and is 20 bytes in length. Immediately after the file header is the second data structure, called a page header. A page consists of a 205 byte page header and space allocated for 1000 data record entries, making a total page size of 20205 bytes. When an IDX file is initially created, it is created with a file header an a single page, making a size of 20225 bytes.

The page header itself consists of some unknown information, but includes a pointer to the next page header (or -1 if it's the last page header), and accounting information about record sizes, number of record slots used and what record slots are allocated.

New pages appear to be allocated on an as needed basis, as the current page in the file is filled. As page sizes are always exactly the same, if the number of pages is known, the exact file size can be calculated. This is useful if the file is being recovered after being deleted, however it does require that the last two page headers be located in fragments that can be pieced together, to create a single portion of the file.

The reason for this is that as the pages are always the same size, the page headers are known to occur at exact intervals in the file. A page header that has a pointer to a subsequent page header can have its own position calculated by subtracting 20205 from the pointer value to gain the position of that page header. This can not be done with the last page header as its pointer value is -1. Hence the previous page header is required to get the pointer value of the start of the last page. Adding 20205 to that file offset gives the exact size of the IDX file.

File fragments containing page headers can be positioned by ordering them by the values of their pointers to the next file. This is useful when reconstructing log files as the exact position of the start of each page can be calculated.

The third data structure is the data records themselves. The data records are arranged in a doubly linked list of valid record entries, with each record storing the position in the IDX file of the previous and next record in the linked list. The other information stored in the record is the status of the entry, the DAT entry number and the file offset within the DAT file of that record. The IDX file can be used to find any and all valid entries in the DAT file, which contains

the logged information. This information can be used to help recreate DAT files from fragments, and also to help recreate IDX files, due to the file offsets that are stored.

The IDX Reader program verified the information as correct, showing that a correct algorithm had been created for extracting IDX information. While this information was not essential to the project, it did provide the following benefits:

1. Assisted with locating records in the DAT file for analysis

2. Understanding of the way information was stored in binary files

3. Verify programming techniques used for extraction of information

4. Provide algorithms that could be used to provide further verification of data extracted from the DAT files

5. Provide a tool to assist in manual analysis of the DAT files.

3. ANALYSIS OF THE DAT FILE

3.1 Dat File Recovery

Dat file recovery should be simple once enough records have been extracted out of the IDX file. The record signatures in figure 4 could be used to locate records and their entry numbers. The extracted linked list entries could then be searched for matching entries, which would have the offset of the DAT entry. The list presented in figure 4 may not be a comprehensive list of all DAT file entries but it Strictz (Strickz, 2002) says that it covers most entry types.

```
Various messages:    E0,23,A3,DB,DF,B8,D1,11,8A,65,00,60,08,71,A3,91
Chat request:
        E1,23,A3,DB,DF,B8,D1,11,8A,65,00,60,08,71,A3,91
File request:
        E2,23,A3,DB,DF,B8,D1,11,8A,65,00,60,08,71,A3,91
My Details:
        E4,23,A3,DB,DF,B8,D1,11,8A,65,00,60,08,71,A3,91
Contact:         E5,23,A3,DB,DF,B8,D1,11,8A,65,00,60,08,71,A3,91
Reminder:
        E6,23,A3,DB,DF,B8,D1,11,8A,65,00,60,08,71,A3,91
Note:         EE,23,A3,DB,DF,B8,D1,11,8A,65,00,60,08,71,A3,91
```

Figure 4: DAT entry signatures (Strickz, 2002)

As different versions of the official Mirabilis ICQ client were released, changes were made to the structure of the log file databases. These changes mean that any program that extracts information from a database has to be able to identify the version of the program that created the database first, and then extract the information and format it according to the version of ICQ that created the file under examination.

Initial information that was downloaded showed numerous versions of the log files, each with differences that needed accounting for when being extracted and formatted. After some analysis, it was found that each record had a record header that is consistent across every version of the databases looked at. This header contained the length of the record, the record number, filing information, a message signature, and what was described in the documentation as a "separator value". The record header is consistent across all versions of the ICQ database. Even the 2003a client database, which has significant differences from all other versions, appears to have this header format.

Another consistency across all versions, 2003b not withstanding, was that the structure of normal text messages was identical. This made it much simpler to implement a system that could extract useful data. As ICQ is primarily a text based messaging system, much of the useful information could be expected to be extracted from these types of messages. It made sense therefore that once a program had been created to extract records that the first records that would be interpreted would be messages. This would allow extraction of text conversations in versions of ICQ up to and including 2003a.

3.2 The Issue of Deleted Log Entries

Initial research indicated that records were probably not deleted from the DAT file. Instead the records in the IDX file that referenced records to be deleted were removed. The freed record space in the DAT file would then be used as required.

This often meant that whole and partial records were left in the DAT file. When the first attempt to create a program to extract DAT records was created, the program simply searched the file for the various signature values, read the record length and copied the data into buffers that was then put into a linked list.

The problem with this was that when signatures from deleted records would be found, often parts of the records had been overwritten, including the start of record with the record length. When run over the test data, when a record was found with a length field that was overwritten to zero length, the program entered an infinite loop. The only to recover from this was to terminate the program.

While this method would have been suitable for databases that contained no deleted data, it was clearly not suitable for databases that contained numerous records that had been deleted over time. It is also not suitable to ignore deleted data as it may contain vital information that could be used. This meant that using the IDX file to recover only valid records was also unsuitable. Another reason for not using the IDX file is that if it was deleted and overwritten it would be unavailable for use. The only useful way to extract the information

would be to locate and identify records and partial records in the DAT file and successfully extract them.

4. HOW THE ENTRIES WERE EXTRACTED

4.1 Creating an Extraction Algorithm

Several ideas were thought up and worked through on paper, modified and refined until one method was found that would be suitable for extracting the log files and a prototype made. First a search function was created that would return the file offset of the next signature in the file, or zero if the end of file was found instead.

The search function was called and then if a signature was found the program entered a loop. A data structure was created to store the record and accounting information about it. The search function was then called to find the next signature. If another signature was found then the length of the current record was calculated and the end position was calculated, if possible. If the previous record possibly overwrote the start of the current record then the end of record was calculated to be the last byte before the start of the next record. The start of the first record was always assumed to be valid.

If the end of the record was calculated to occur after the start of the next record then the two records were deemed to overlap. The end of the current record and the start of the next record were recorded as "DIRTY" or "OFFSET" and the number of overlapping bytes was recorded with each record.

The length of the record was calculated and that much data read into the data structure. Accounting information about the status of the end of the record was also added and then the record structure added the end of the linked list. Information obtained about the next signature was stored in local variables. The loop then iterated. After the next record data structure was created, the information would be copied to the new data structure. The loop exited when no new records were found. Once the prototype was able to extract all records and partial records, it was ported to C++ and record structures were added to for validation and output of data. Figure 5 shows the extraction algorithm used.
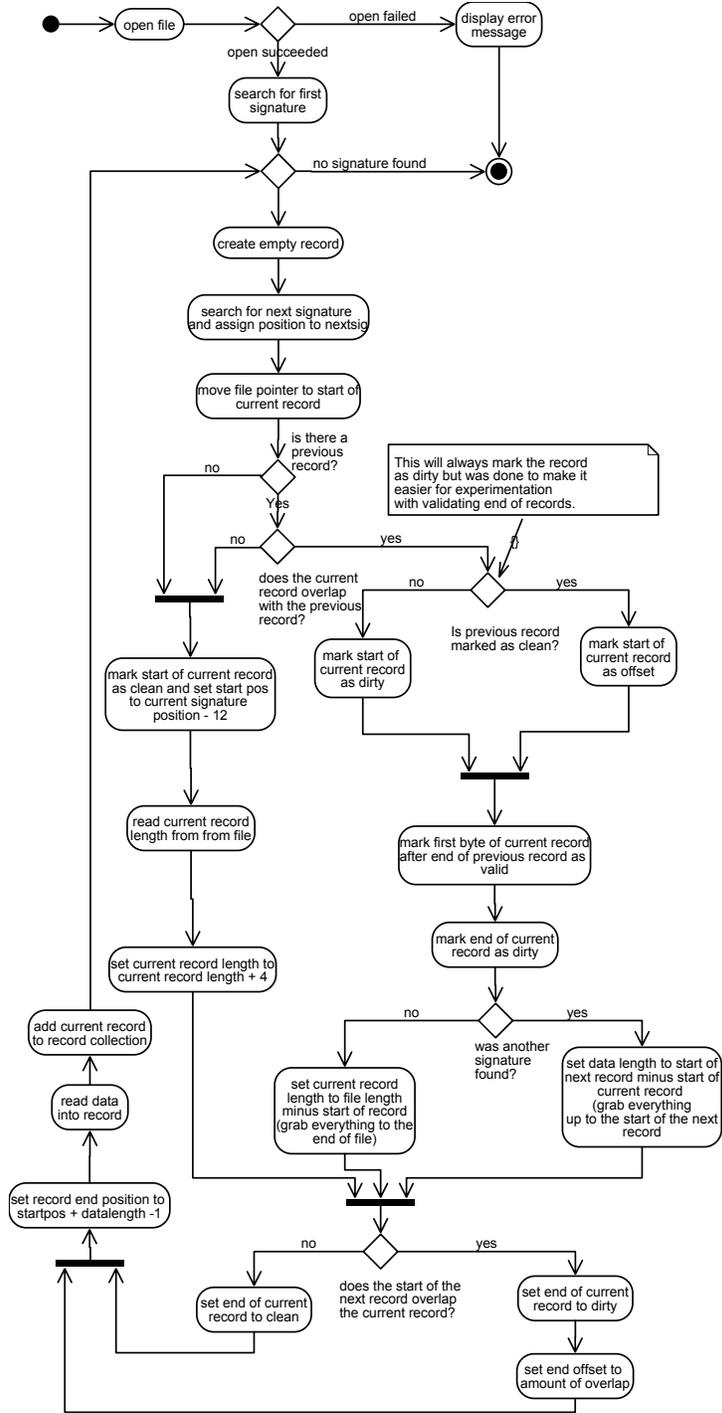
Figure 5: Extraction algorithm

60

4.2 Identifying the ICQ Version

Initially it was thought that extensive examination of the record structures would be required to identify the version of ICQ being used. After some analysis it was noted that for the versions of ICQ listed in the downloaded information, the separator value changed with each version. The decimal notation of the separator value corresponded with the version number that created the log. What this meant was that a single record could be located and from that value the exact version of the ICQ client that created the log could be identified.

4.2 Identification and Verification of Entries

Identification of individual record types is also quite simple. The signature values are all 16 bytes in length. Most signatures differ only in the first byte with the last 15 bytes being identical. The first byte however, is unique to each record type and provides an excellent method for identifying the type of record being extracted.

As information sorted out about the data that was contained in each data structure, a class hierarchy was created that had its root class as containing the header structure only. From there a level of classes was created that contained information common to all versions of ICQ for a type of record. The third layer generally contained information specific to individual record types.

This allows the manipulation of records as a single record type, as a type of record such as "Message", or as a specific type of message, depending on what was required from the data structure.

Each data structure inherits a method called verify from the base class which can be overwritten. Verify is designed to verify the information stored specifically by that class if possible. Each class would contain information specific to that class and would be designed to verify the validity of that data. In the initial test version of this program, there was no validation of entries as to test the basic concepts only entries verified as being valid were used. In later implementations of the program validation methods would be identified and added to the program.

5. TESTING

Once the classes were created and a library of functions to do endian conversions created, outputting the records was quite simple. Streaming operators were created and the output was simply streamed to the console. Once a significant amount of work had been completed, a chat session was held and a small amount of test data obtained from it. The version of ICQ used was 2003a, the most recent version that can be analysed with this version of the extraction program. Using the log files that were obtained, three types of analysis was performed. These were manual, using the ICQ program itself to

output the history of the conversation, and using the extraction program that was created in this project.

Initial results of this program are that the program is extremely accurate in the limited output that it can achieve with only one limitation. The analysis of the timestamps does not account for the time zone, which for Perth is GMT +08:00

So far a fairly extensive amount of analysis has been completed allowing for the creation of this program. The program itself has the following limitations:

1. Areas of possible concern for the algorithm have been identified and need to be investigated. This is further discussed in the next section on future work.

2. Only the message structures being identified and output.

3. There are no options to adjust time stamps for time zones or discrepancies in time. The program does adjust for time zones.

4. Only complete entries that are marked as "CLEAN" are used. There is no validation of the entries that would allow the other records to be used.

5. The extracted data records are not immutable, which means that they could be accidentally altered.

6. The program does not sort by UIN number or time, so while all the messages are output they are not sorted into conversations, which must be done manually.

7. The current data structures do not allow for the 2003b version of the ICQ client to be analysed. This is because the database is now distributed over a number files, signatures have changed, and the structure of the messages have changed.

While these limitations reduce the usefulness of the program somewhat in it's current form, it still has great usefulness in reducing the amount of time needed to extract useful information, which would then need to be manually organised.

6. FUTURE WORK

While records are being extracted from the databases, there is a large amount of work to be done. The first task will be to address the limitations that have already been mentioned above. The concern with the extraction algorithm is that although the current algorithm performs flawlessly, it depends on the first record being valid. Also the algorithm has yet to be tested with multiple "DIRTY" records one after the other. Further refinements may be required to make the algorithm more robust.

Another purpose for strengthening this algorithm is that it could be used to analyse partial database files where the first record in a file fragment is not

"CLEAN". It is possible that once this project is completed, as second project or an extension of the current project is undertaken to create a program that locates database file fragments and reconstructs a database file. This is not a concern for this project, however it would be useful for a forensic analyst to have such a capability and this could be considered at a later date.

There are a number of other data structures that need to be added to the program. These data structures all require that the version of the program be known as the data structures to be used are specific to that version of the database. Thus work needs to be done to properly implement objects for each version of the database that use the required data structures. These objects could be used to hold global database information which could be used for various purposes. What this data is has yet to be identified. However, validation routines could be expected to use this information.

It is also expected that validation routines will increase the amount of accounting information that a record will need to keep about its associated data, such as previous and next records. Another concern is that only records that have a complete signature are found. In principle it is possible that a record could exist that contains useable data, however does not have a valid signature as it has been overwritten. It would be useful to extend the extraction algorithm to detect partial records without a complete signature. In this event, it would require the validation routines to be able to identify a record type from a partial record. Therefore research is required to determine how much of a record type is required before its type can be determined.

7. REFERENCES

Anonymous. (2002) Computer-aided crime faces computer-aided forensics. Last Update September 18, 2002, Retrieved 16th April, 2005, from http://www.info.gov.hk/gia/general/200209/18/0918158.htm

Hitu. (2002). IcqHR (Version 1.8).

Mosnews. (2005), U.S. Cyber-Crime Unit Focuses on Russian Hackers. Retrieved 14th April, 2005, from

http://www.mosnews.com/news/2005/04/05/compcrime.shtml

Poulsen, K. (2005), *Hacker penetrates T- Mobile Systems*, Retrieved 14th April, 2005, from http://www.crimeresearch.org/news/12.01.2005/892/

Soeder, D. (2000), Icqnewdb.txt. Last Update 19th April, 2000, Retrieved April 6, 2005

Strickz. (2002), ICQ Db Specs. Last Updated 8th July, 2002, Retrieved April 5, 2005, from http://cvs.sourceforge.net/viewcvs.py/miranda-icq/Plugins/import/docs/import-ICQ_Db_Specs.txt