Forensic Relative Strength Scoring: ASCII and Entropy Scoring

Matthew M. Shannon
Principal, Agile Risk Management

"There is safety in numbers..."
-Anonymous

Abstract

This paper is the result of an investigation into applying statistical tools and methodologies to the discovery of digital evidence. Multiple statistical methods were reviewed; the two most useful are presented here. It is important to note that this paper represents an inquiry into the value of applied mathematical analysis to digital forensics investigations. Readers are encouraged to explore the concepts and make use of the tools presented here, in the hope that a synergy can be developed and concepts can be expanded to meet future challenges. In addition, this paper contains practical examples using modified Sleuthkit[1] tools containing the proposed statistical measurements.

Introduction

Numbers, when properly defined and calculated, can provide us with staggering insights into the very make up of our world and our interactions with it. Statistics dominate much of the commercial world. We review the average passing yards and sacks per game prior to our team's big weekend showdown, we analyze price to earnings ratios and buy vs. sell percentages prior to a sizable investment, we even use unit pricing proportions to decide which brand of soap will provide the most value. In all of these actions, numbers, and more importantly, statistics, play a much needed role. How accurate would our decisions be, financial or otherwise, without simple statistics and mathematics?

The need for statistics and numerical analysis rises again when we enter into the world of digital forensics. We as forensic practitioners are often faced with enormous amounts of data, sub-optimal searching strategies, and incomplete information. For example:

- Searching for obfuscated documents, text hidden within system files, or encrypted data.
    - Many times encrypted data is made to look just like established system files or arbitrary junk. How do we locate this information accurately? How do we detect system files that have been

[1] The Sleuthkit Open Source Forensics toolkit, http://www.sleuthkit.org

doctored, and contain text critical to our investigation?

- Analyzing large amorphous files, such as pagefiles, temp files, and swap files.
    - These files are often hundreds of megabytes in size, and contain seemingly random bits of information, often intermixed among useful text data.

This leaves the forensics practitioner in a difficult position. How can the amount of time-consuming analysis performed be reduced, and how can valuable information be located in a reasonable amount of time?

Proposed Solution

As alluded to earlier, statistical tools play a valuable role in reducing analysis time and increasing the accuracy of decisions. Therefore, in order to reduce the amount of data requiring analysis and to pinpoint specifically valuable information, the author has developed a multi-factor statistical scoring system. This system utilizes three different statistical scores, each presenting different information about the inputted data point, allowing the examiner to limit his focus to specific data units based on the specific target very quickly. In addition, this system must be scalable. Calculations must be expedient as well as accurate, and they must be applicable to all the different data units that could be presented.

In order to address these issues the techniques and formulas developed and perfected by others were reviewed and adjusted for applicability to the task at hand. In the end the following two statistical measures were selected to comprise the initial “Forensic Relative Strength Scoring system,” or FRSS. They are:

- ASCII Proportionality
    - In a given data unit, what proportion of the data unit is humanly readable ASCII code? This may assist in keyword searching, as data units having little or no humanly readable ASCII could be removed from the set of search candidates.
- Entropy
    - Entropy is loosely defined as the relative randomness of a given data unit. In this study, this score will help identify zip, compressed, or even encrypted files, as their relative randomness will be very high.

Forensic Relative Strength Scoring (FRSS)

ASCII Proportionality

With all things being equal, the probability of discerning valuable textual information from a given data unit rises based on the portion of ASCII to non-ASCII values it contains. In short, when searching for natural language keywords, those data units containing more text in relation to their overall size have a higher probability of containing useful information. In order to better illustrate this example, the following are outputs from a modified version of the popular Sleuthkit open source forensic toolkit. The istat and fls commands shown here provide information about inodes, or files. In this instance ASCII proportionality, or a percentage of ASCII printable characters comprising the input has been added, as a measure for scoring the outputs.

```
[root@silentpower bin]# ./istat-shannon -f fat ../../test.img 8
Directory Entry: 8
Allocated
DOS Mode: File
size: 9349
num of links: 1
Name: lxrh.dll <----------------------------------System File, low chance for printable
                                                  characters.

Directory Entry Times:
Written:        Thu Sep 30 07:20:00 1993
Accessed:       Wed Dec 31 19:00:00 1969
Created:        Wed Dec 31 19:00:00 1969

Sectors:
1365 1366 1367 1368 1369 1370 1371 1372
.....lines removed.....
1421 1422 1423 1424 1425 1426 1427 1428

Ascii Score:0.009627<---------------------------ASCII Score shows this to be true,
                                                        as < 1% of the file is printable
                                                  ASCII.

[root@silentpower bin]# ./istat-shannon -f fat ../../test.img 33
Directory Entry: 33
Allocated
DOS Mode: File
size: 501
num of links: 1
Name: AUTOEXEC.BAT<--------------------Batch file, most likely containing
                                                  text.
Directory Entry Times:
Written:        Mon Apr 15 17:20:04 1996
Accessed:       Wed Dec 31 19:00:00 1969
Created:        Wed Dec 31 19:00:00 1969
```

```
Sectors:
1493 1494 1495 1496 1497 1498 1499 1500
.....lines removed.....
1549 1550 1551 1552 1553 1554 1555 1556

Ascii Score:0.928144<------------------------ASCII Score proves this to be true, as
                                              the file contains roughly 92% text.
```

The above files were chosen to clearly show the stark differences between percentages of printable ASCII by file type. This becomes more useful when browsing directories.

```
[root@silentpower bin]# ./fls2 -la -f fat ../../test.img 32
   SECTOR          FILENAME                        ASCII_SCR      SIZE
r/r 9529861:    LISTMGR.DLL                 0.306958        17504    0       0
r/r 9529862:    MCONTROL.EXE                0.319503        803580   0       0
r/r 9529863:    MCONTROL.HLP                0.236096        232778   0       0
r/r 9529864:    SETUP.BMP                   0.182188        20918    0       0
r/r 9529865:    SETUP.EXE                   0.309251        102496   0       0
r/r 9529866:    SETUP.INF                   0.905817        1083     0       0
.....lines removed.....
```

The SETUP.INF file shows the largest ASCII score, which mirrors what is expected, as .INF files are typically system files containing textual windows driver information. While in the case above nothing novel was discovered, imagine the same query resulted in the following values:

```
[root@silentpower bin]# ./fls2 -la -f fat ../../test.img 32
   SECTOR          FILENAME                        ASCII_SCR      SIZE
.....lines removed.....
r/r 9529864:    SETUP.BMP                   0.982188        20918    0       0
r/r 9529865:    SETUP.EXE                   0.309251        102496   0       0
r/r 9529866:    SETUP.INF                   0.905817        1083     0       0
```

Could this BMP file be hiding valuable information? 98% is a large proportion of printable text relative to the size of the file, further review would be warranted.

Entropy

Borrowing from Claude Shannon's groundbreaking work in Information Theory[2], Entropy was chosen as one of the constructors of FRSS. To summarize one of Shannon's concepts, Entropy is a measure of the information density or compression state of a given unit of data. The more a given unit can be compressed, the lower the Entropy value; the less a given unit can be compressed, the higher the Entropy value. For example, Bitmap Image files and ASCII text files are typically highly compressible. Therefore, they have a low Entropy value. Encrypted data is typically not compressible, and, therefore, possesses a high Entropy value.

[2]Claude Shannon, *A Mathematical Theory of Communication*, http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html

In the example, Entropy will be used as an indicator of the variability of bits per a given byte. Since each character in a given data unit consists of one byte, this will provide an indication as to the variability of given characters. This value, expressed as the number of bits per byte of a given data unit, essentially provides an indication as to the information density or compressibility of the data unit.

In order to better illustrate the point, a simple Entropy calculation tool, "ent," developed by John Walker [3] has been used. The following three sets of information will be submitted into ent and the outcomes recorded.

Ex. Three different input files were developed for this example. Each input file contained text; however, the consistency of the text differed as follows:

identical.test
Consisting of 24,767 bytes of the identical letter, 'a'.

natural.test
Consisting of 24,767 bytes of naturally occurring English text, taken from Jack London's Call of the Wild, provided by Project Gutenberg.

random.test
Consisting of 24,767 bytes of random values from provided by the Linux /dev/random device.

The resulting Entropy scores below show the sizable difference between random, identical, and naturally occurring text.

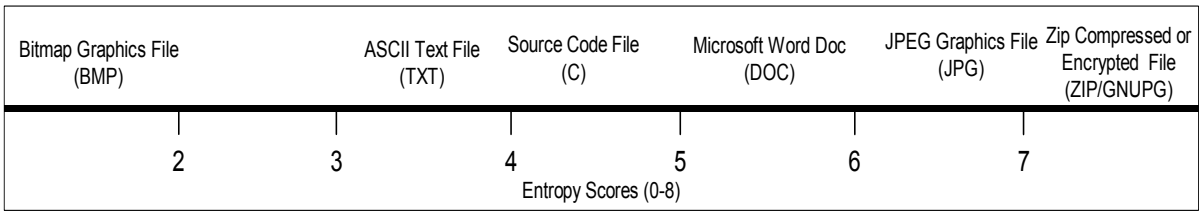
```
[mshannon@silentpower demo]$ ../ent indentical.test
Entropy = 0.072742 bits per byte.

[mshannon@silentpower demo]$ ../ent random.test
Entropy = 7.992178 bits per byte.

[mshannon@silentpower demo]$ ../ent natural.test
Entropy = 4.415043 bits per byte.
```

As nature would have it, the ASCII Text files possess roughly 4~5 bits of entropy per byte. While this value will fluctuate slightly based on the input text, it is a reasonable measure, useful in determining the relative make up of a given information unit. The following chart shows an approximate spectrum of Entropy scores derived by FRSS.

[3] John Walker, *Pseudorandom Numbers*, http://www.fourmilab.ch/random



Spectrum of Approximate Entropy Calculations

Therefore, using the concepts developed by John Walker, the istat and dls commands have again been modified, this time by adding an Entropy score in addition to the ASCII score.

```
[root@silentpower bin]# ./istat -x -f fat ../../test.img 33
Directory Entry: 33
Allocated
DOS Mode: File
size: 501
num of links: 1
Name: AUTOEXEC.BAT

Directory Entry Times:
Written:        Mon Apr 15 17:20:04 1996
Accessed:       Wed Dec 31 19:00:00 1969
Created:        Wed Dec 31 19:00:00 1969

Sectors:
1493 1494 1495 1496 1497 1498 1499 1500
.....lines removed.....
1549 1550 1551 1552 1553 1554 1555 1556

Entropy Score:5.008025
Ascii Score:0.928144
```

In the above case not only is a high ASCII score detected, but a relatively moderate Entropy score. What does this indicate? That the document falls within the 3-5 bits of entropy per byte, and contains roughly 93% printable text. This indicates a strong match for a humanly readable file.

Furthermore, as with ASCII scoring above, Entropy scoring becomes more valuable when provided in a list while browsing directories.

```
[root@silentpower bin]# ./fls -x -l -f fat ../../test.img 32
    SECTOR          FILENAME                ENTROPY       ASCII_SCR      SIZE
r/r 9529861:    LISTMGR.DLL       6.252049        0.306958        17504
r/r 9529862:    MCONTROL.EXE    5.742111        0.319503        803580
r/r 9529863:    MCONTROL.HLP     6.777656        0.236096        232778
r/r 9529864:    SETUP.BMP          1.720249        0.182188        20918
r/r 9529865:    SETUP.EXE          6.537036        0.309251        102496
r/r 9529866:    SETUP.INF          5.438305        0.905817       1083
.....lines removed.....
```

What does this indicate? Combining the fact that Entropy is a measure of randomness

with the characteristics of Bitmap files, it is evident that Bitmap files are non-compressed (low entropy), and possess relatively little printable ASCII.

Now what would happen if the results were as follows instead:

```
[root@silentpower bin]# ./fls –x -l -f fat ../../test.img 32
   SECTOR          FILENAME                ENTROPY       ASCII_SCR      SIZE
r/r 9529861:   LISTMGR.DLL        6.252049          0.306958         17504
r/r 9529862:   MCONTROL.EXE    5.742111          0.319503         803580
r/r 9529863:   MCONTROL.HLP     6.777656          0.236096         232778
r/r 9529864:   SETUP.BMP           7.920249          0.782188         20918
r/r 9529865:   SETUP.EXE           6.537036          0.309251         102496
r/r 9529866:   SETUP.INF           5.438305          0.905817       1083
.....lines removed.....
```

Entropy is a measure of randomness or information density, and since compressed or encrypted files possess the largest levels of Entropy, it is reasonable to believe that the SETUP.BMP file may not truly be a bitmap file, and may be an encrypted or compressed data file. Regardless, it immediately warrants further investigation. Entropy scoring highlights potential file information quickly, even in cases where known file patterns, or "File Magic," fail.

In addition to whole file scoring, understanding the statistical scoring of individual sectors of large amorphous file is also valuable.

By applying the same techniques outlined about, useful information can be obtained about individual sectors within files such as pagefile.sys, swap files, and temp files. With this in mind, the Sleuthkit istat command was modified again, this time by adding per sector scoring values.

```
[[root@silentpower bin]# ./istat -x -f ntfs ../../test.img 24 | more
MFT Entry: 24
Sequence: 3
Allocated
UID: 0
DOS Mode: File, Hidden
Size: 402653184
Links: 1
Name: pagefile.sys

$STANDARD_INFORMATION Times:
Created:        Tue Sep 23 08:22:08 2003
File Modified:  Fri Jan 16 09:33:56 2004
MFT Modified:   Fri Jan 16 09:33:56 2004
Accessed:       Fri Jan 16 09:33:56 2004

$FILE_NAME Times:
Created:        Tue Sep 23 08:22:08 2003
File Modified:  Fri Sep 26 11:02:17 2003
MFT Modified:   Fri Sep 26 11:02:17 2003
Accessed:       Fri Sep 26 11:02:17 2003

Attributes:
Type: $STANDARD_INFORMATION (16-0)   Name: N/A   Resident   size: 72
```

```
Type: $FILE_NAME (48-2)   Name: N/A   Resident   size: 90
Type: $DATA (128-1)   Name: $Data   Non-Resident   size: 402653184
Sectors:
5152948 5152949 5152950 5152951 5152952 5152953 5152954 5152955
5152956 5152957 5152958 5152959 5152960 5152961 5152962 5152963
5251244 5251245 5251246 5251247 5251248 5251249 5251250 5251251
Scoring:
Sector:5152948  Entropy:4.137966      ASCII:0.981107
Sector:5152949  Entropy:5.692130      ASCII:0.880518
Sector:5152950  Entropy:2.688838      ASCII:0.281250
Sector:5152951  Entropy:2.096043      ASCII:0.276855
Sector:5152952  Entropy:1.748843      ASCII:0.302002
Sector:5152953  Entropy:1.484269      ASCII:0.264160
Sector:5152954  Entropy:1.927856      ASCII:0.111768
Sector:5152955  Entropy:1.166364      ASCII:0.318848
Sector:5152956  Entropy:1.060080      ASCII:0.287354
Sector:5152957  Entropy:0.970360      ASCII:0.312744
.....lines removed.....
Sector:5251251  Entropy:0.000242      ASCII:0.357910

Entropy Score:5.623937
Ascii Score:0.233373
```

In the above case the overall pagefile.sys has an entropy of 5.6, while the individual sectors have lower entropy. In fact, the sectors differ greatly by entropy. This means that the sectors of most interest are 5152948 and 5152949, containing an Entropy closely linked to text and office documents, between 3~5 bits per byte, plus roughly 98%~88% printable text. This indicates a possible match for a humanly readable sector.

Future Directions

The tools and techniques developed above provide a sizable amount of new and interesting information, complimenting the existing forensics process. There are, however, additional directions and considerations discovered during development of FRSS, which will be considered in future revisions. These include N-Gram Scoring.

Armed with an understanding of the data unit's ASCII proportion, as well as a numerical indication as to its relative entropy, all that remains is to mathematically determine whether or not the given data unit resembles written English. It is here where established systems for language identification come into play. Of these, N-Gram based text categorization provides considerable accuracy with minimal computational overhead. The N-Gram categorization system being used is the distance-matching algorithm developed by William B. Cavner and John M. Trenkle in “N-Gram-Based Text Categorization”[4].

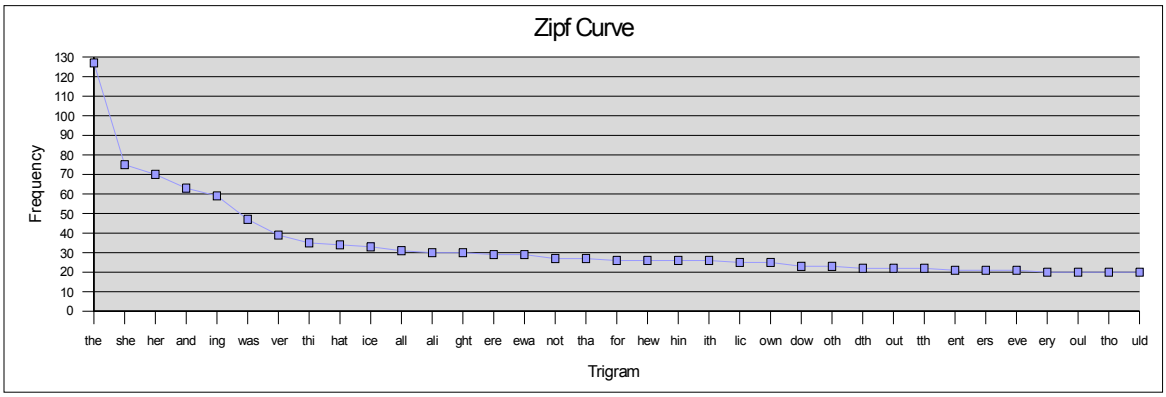
[4]William B. Cavner & John M. Trenkle, *N-Gram-Based Text Categorization*, http://www.nonlineardynamics.com/trenkle/papers/sdair-94-bc.ps.gz

To summarize, Cavner & Trenkle proposed comparing the ranked frequency of occurrence of N length character sequences to the average ranked frequency of occurrence of those same N length character sequences in a given language. These character sequences are called N-Grams and consist of N character continuous slices of a longer string. Cavner & Trenkle proposed that the difference in ranking between the most frequently occurring N-Grams would remain small for two texts written in the same language.

To our benefit, human languages always have some N-Grams that appear with more frequency than others. This idea has been clearly articulated by Zipf's law[5], restated by Cavner & Trenkle as:

The nth most common word in human language text occurs with a frequency inversely proportional to n.

In its most basic sense, Zipf's law shows that there is a set of N-Grams which dominate most other N-Grams in terms of frequency of use for a given language. This law forms the basis for most N-Gram based language identification algorithms. The following is a chart of Zipf's Law.



However, unlike language identification systems, the key concern here remains separating English text from binary data or relatively non-random text. While current systems remain focused on language comparison, they are not well developed for distinguishing text for relative random values in a manner easily adaptable to the forensic analysis process. The author is currently developing a N-Gram scoring system more amenable to the forensics analysis process and anticipates its appearance in future versions of FRSS.

Concerns

As with any statistical system, concerns as to weighing and scoring are certainly warranted. Many of the anticipated concerns have been addressed below.

[5]GK Zipf, *Selective Studies and the Principle of Relative Frequency in Language (*1932)

However, as the system evolves and grows, new concerns may become apparent that were not considered here.

- How do I obtain the code and software detailed in this paper?
  - The software and methodology used to generate Entropy and ASCII Proportionality scores is available as a patch to the Sleuthkit 1.69 package at our website(http://www.agilerm.net/), or in the appendix of this document. It is recommended that others generate new results and explore theories presented here.

- Why did you use two different scoring mechanisms?
  - Each scoring mechanism provides complimentary information. For instance, a data unit with a high ASCII score may be appealing, however if it also has a very low Entropy, it may be relatively useless repeating characters.

- If I depend on FRSS isn't there a chance I will miss important information?
  - Yes. FRSS is intended to be a guide, not an absolute decision maker. FRSS was developed to provide additional information to help with the sorting and analysis process. Just as one would not depend on statistical scores alone to choose an investment vehicle, one would not depend on FRSS alone to locate information.

About the Author

Matthew Shannon has over five years of professional information security experience in private industry, including KPMG LLP, ExxonMobil, and United Technologies. Mr. Shannon has been the lead investigator on numerous computer forensics engagements, including intellectual property theft and employment law.

Mr. Shannon graduated cum laude from The University of Florida in Decision and Information Sciences (BSBA) in 1999. He is a well received speaker and author, having presented at the DEFCON 11 Information Security Conference in Las Vegas, Nevada, in addition to a recent request to instruct the US Secret Service on specific digital forensics issues . Mr. Shannon is a member in good standing of ISSA. In addition, he holds numerous professional information technology certifications, and has been a contributor to multiple open source information

security projects, including "The Sleuthkit" and "The Honeynet Project."

More information on Mr. Shannon, as well as Agile Risk Management, is available at www.agilerm.net. Further questions or comments should be sent to mshannon@agilerm.net.

References

Carrier, Brian et all, Sleuthkit (Formerly TASK) Forensic Software Suite; http://www.sleuthkit.org

Cavner, William B & Trenkle, John M, N-Gram-Based Text Categorization; April 1994; Electronic version retrieved 12th February from http://www.nonlineardynamics.com/trenkle/papers/sdair-94-bc.ps.gz

Shannon Claude, A Mathematical Theory of Communication; February 2, 1998; Electronic version retrieved 12th February from http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html

Walker John , Pseudorandom Numbers; Electronic version retrieved 12th February from http://www.fourmilab.ch/random

Zipf, GK , Selective Studies and the Principle of Relative Frequency in Language (1932)

Appendix A: FRSS-Sleuthkit Patch

The following patch was developed for Sleuthkit 1.69

```
diff -aurN sleuthkit-1.69/src/fstools/fls.c sleuthkit-1.69-frss/src/fstools/fls.c
--- sleuthkit-1.69/src/fstools/fls.c 2004-01-06 17:50:52.000000000 -0500
+++ sleuthkit-1.69-frss/src/fstools/fls.c    2004-05-20 14:15:51.000000000 -0400
@@ -58,7 +58,7 @@


 void usage(char *myProg) {
-       printf("usage: %s [-adDFlpruvV] [-f fstype] [-m dir/] [-z ZONE] [-s seconds] image
[inode]\n",
+       printf("usage: %s [-adDFlxpruvV] [-f fstype] [-m dir/] [-z ZONE] [-s seconds]
image [inode]\n",
          myProg);
      printf("\tIf [inode] is not given, the root directory is used\n");
       printf("\t-a: Display \".\" and \"..\" entries\n");
@@ -68,6 +68,7 @@
        printf("\t-l: Display long version (like ls -l)\n");
        printf("\t-m: Display output in mactime input format with\n");
        printf("\t       dir/ as the actual mount point of the image\n");
+       printf("\t-x: Display FRSS Scoring\n");
        printf("\t-p: Display full path for each file\n");
        printf("\t-r: Recurse on directory entries\n");
        printf("\t-u: Display undeleted entries only\n");
@@ -273,7 +274,7 @@

        localFlags =  LCL_DIR | LCL_FILE;

-       while ((ch = getopt(argc, argv, "adDf:Fm:lprs:uvVz:")) > 0) {
+       while ((ch = getopt(argc, argv, "adDf:Fm:lxprs:uvVz:")) > 0) {
                switch (ch) {
                case '?':
                default:
@@ -298,6 +299,9 @@
                case 'l':
                        localFlags |= LCL_LONG;
                        break;
+               case 'x':
+                       ent_report =1;
+                       break;
                case 'm':
                        localFlags |= LCL_MAC;
                        macpre = optarg;
diff -aurN sleuthkit-1.69/src/fstools/frss.c sleuthkit-1.69-frss/src/fstools/frss.c
--- sleuthkit-1.69/src/fstools/frss.c       1969-12-31 19:00:00.000000000 -0500
+++ sleuthkit-1.69-frss/src/fstools/frss.c  2004-05-23 14:46:48.000000000 -0400
@@ -0,0 +1,233 @@
+/*
+** frss
+** The Sleuth Kit
+**
+** Given a block number, compute the Forensic Relative Strength Scoring
+** values of the block, return them to calling function.
+**
+** Matthew Shannon [mshannon@agilerm.net]
+** Copyright (c) 2004 Matthew Shannon. All rights reserved.
+**
+** Brian Carrier [carrier@sleuthkit.org]
+** Copyright (c) 2003 Brian Carrier.  All rights reserved
+**
+** TASK
+** Copyright (c) 2002 Brian Carrier, @stake Inc.  All rights reserved
+**
+** TCTUTILs
```

```
+** Brian Carrier [carrier@cerias.purdue.edu]
+** Copyright (c) 2001 Brian Carrier.  All rights reserved
+**
+** Redistribution and use in source and binary forms, with or without
+** modification, are permitted provided that the following conditions are
+** met:
+**
+** 1. Redistributions of source code must retain the above copyright notice,
+**    this list of conditions and the following disclaimer.
+** 2. Redistributions in binary form must reproduce the above copyright
+**    notice, this list of conditions and the following disclaimer in the
+**    documentation and/or other materials provided with the distribution.
+** 3. The name of the author may not be used to endorse or promote
+**    products derived from this software without specific prior written
+**    permission.
+**
+**
+** THIS SOFTWARE IS NOT AFFILIATED WITH PURDUE UNIVERSITY OR THE CENTER FOR
+** EDUCATION IN INFORMATION ASSURANCE AND SECURITY (CERIAS) AND THEY BEAR
+** NO RESPONSIBILITY FOR ITS USE OR MISUSE.
+**
+**
+** THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
+** WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
+** MERCHANTABILITY AND FITNESS FOR ANY PARTICULAR PURPOSE.
+**
+** IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
+** INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
+** (INCLUDING, BUT NOT LIMITED TO, LOSS OF USE, DATA, OR PROFITS OR
+** BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
+** WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
+** OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
+** ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
+**
+*/
+
+#include "fs_tools.h"
+#include "error.h"
+#ifndef __FRSS_H__
+#include "frss.h"
+#endif
+#include <ctype.h>
+
+int char2num(int c)
+{
+    if( isalpha(c) )
+        return tolower(c) - 'a' + 1;
+    else
+        return 0;
+}
+
+int num2char(int c)
+{
+    if( c == 0)
+        return ' ';
+    else
+        return c - 1 + 'a';
+}
+
+/*  LOG2  --  Calculate log to the base 2, entropy calculation */
+
+double log2(double x)
+{
+    return log2of10 * log10(x);
+}
+
+/* Clear out all variables */
+
```

```
+void clear_vars(){
+       int i;
+       for (i=0;i<256;i++){
+               ccount[i]=0;
+               prob[i]=0;
+       }
+       totala=0;
+       totalc=0;
+       totalsize=0;
+}
+
+// Calculate Entropy Values
+
+double ent_calc (){
+
+       double ent =0;
+       int i;
+       for (i = 0; i < 256; i++) {
+        prob[i] = (double) ccount[i] / totalc;
+
+               }
+    /* Calculate entropy */
+
+    for (i = 0; i < 256; i++) {
+       if (prob[i] > 0.0) {
+                       ent += prob[i] * log2(1 / prob[i]);
+       }
+    }
+return ent;
+}
+
+double ent_calc_block (){
+
+       double ent =0;
+       int i;
+       for (i = 0; i < 256; i++) {
+        prob[i] = (double) bccount[i] / totalc;
+
+               }
+    /* Calculate entropy */
+
+    for (i = 0; i < 256; i++) {
+       if (prob[i] > 0.0) {
+                       ent += prob[i] * log2(1 / prob[i]);
+       }
+    }
+return ent;
+}
+
+
+float ascii_calc(){
+       float ascii = 0;
+       ascii = (float) totala/totalsize;
+       return ascii;
+}
+float ascii_calc_inode(int size){
+       float ascii = 0;
+       ascii = (float) totala/size;
+       return ascii;
+}
+float ascii_calc_block(int size, int total){
+               float ascii = 0;
+               ascii = (float) total/size;
+               return ascii;
+}
+float ascii_score_return(){
+               printf("%f\n",ascii_score);
+               return ascii_score;
```

```
+}
+
+
+u_int8_t frss_inode_scoring(FS_INFO *FS, DADDR_T addr, char *buf, int size,
+  int flags, char *ptr){
+       void *h;
+       int iIdx;
+       unsigned char ocb;
+       int oc;
+
+        if (size == 0)
+               return WALK_CONT;
+               for (iIdx = 0; iIdx < size; iIdx++) {
+
+                       oc = (int) buf[iIdx];
+                       ocb = (unsigned char) oc;
+                       totalc++;
+                       ccount[ocb]++;
+               }
+
+       /* Calculate the ASCII Printable Score */
+
+               for (iIdx = 0; iIdx < size; iIdx++) {
+                       oc = (int) buf[iIdx];
+                       ocb = (unsigned char) oc;
+
+                       if (isprint(ocb)){
+                               totala++;
+                       }
+               }
+       totalsize += size;
+
+       return WALK_CONT;
+}
+
+
+
+u_int8_t frss_sector_scoring(FS_INFO *FS, DADDR_T addr, char *buf, int size,
+  int flags, char *ptr){
+       int i,iIdx,total;
+       total =0;
+       unsigned char ocb;
+       int oc;
+
+         for (i=0;i<256;i++){
+               bccount[i]=0;
+         }
+        if (size == 0){
+               return WALK_CONT;
+        }
+               for (iIdx = 0; iIdx < size; iIdx++) {
+
+                       oc = (int) buf[iIdx];
+                       ocb = (unsigned char) oc;
+                       totalc++;
+                       ccount[ocb]++;
+                       bccount[ocb]++;
+               }
+
+       /* Calculate the ASCII Printable Score */
+
+               for (iIdx = 0; iIdx < size; iIdx++) {
+                       oc = (int) buf[iIdx];
+                       ocb = (unsigned char) oc;
+
+                       if (isprint(ocb)){
+                               totala++;
+                               total++;
```

```
+                     }
+             }
+
+       ent_score = ent_calc();
+       float ent_score_sm;
+       ent_score_sm = ent_calc_block();
+       float ascii_score_sm;
+       ascii_score_sm = ascii_calc_block(size,total);
+       ascii_score = ascii_calc_inode(totalc);
+       printf("Sector:%i\tEntropy:%f\tASCII:%f\n",addr,ent_score_sm,ascii_score_sm);
+return WALK_CONT;
+       }
diff -aurN sleuthkit-1.69/src/fstools/frss.h sleuthkit-1.69-frss/src/fstools/frss.h
--- sleuthkit-1.69/src/fstools/frss.h       1969-12-31 19:00:00.000000000 -0500
+++ sleuthkit-1.69-frss/src/fstools/frss.h  2004-05-19 21:58:59.000000000 -0400
@@ -0,0 +1,55 @@
+/*
+** FRSS Functions and Datatypes
+**
+** Matthew Shannon [mshannon@agilerm.net]
+** Copyright (c) 2004 Matthew Shannon. All Rights Reserved
+**
+** Based on code for entropy calculations designed and implemented by John Walker May
1995
+**
+** Brian Carrier [carrier@sleuthkit.org]
+** Copyright (c) 2003 Brian Carrier.  All rights reserved
+**
+** TASK
+** Copyright (c) 2002 @stake Inc.  All rights reserved
+**
+** Copyright (c) 1997,1998,1999, International Business Machines
+** Corporation and others. All Rights Reserved.
+*/
+#ifndef __FRSS_H__
+#define __FRSS_H__
+
+#include <stdio.h>
+#include <string.h>
+#include <ctype.h>
+#include <math.h>
+#include <openssl/evp.h>
+
+
+#define log2of10 3.32192809488736234787
+static long ccount[256],            /* Bins to count occurrences of values */
+           totalc = 0,                      /* Total bytes counted */
+               totala = 0,
+               totalsize = 0,
+               bccount[256];
+
+static double prob[256];            /* Probabilities per bin for entropy */
+
+/* FRSS Scoring Values ********************************************/
+       float ent_score;
+       float ngram_score;
+       float ascii_score;
+/*****************************************************************/
+
+extern int char2num(int);
+extern int num2char(int);
+extern double log2(double);
+extern void clear_vars();
+extern double ent_calc ();
+extern double ent_calc_block ();
+extern float ascii_calc();
+extern float ascii_calc_inode();
+extern float ascii_calc_block(int, int);
```

```
+u_int8_t frss_inode_scoring(FS_INFO *, DADDR_T , char *, int , int , char *);
+u_int8_t frss_sector_scoring(FS_INFO *, DADDR_T, char *, int , int , char *);
+
+#endif
diff -aurN sleuthkit-1.69/src/fstools/fs_dent.c sleuthkit-1.69-frss/src/fstools/fs_dent.c
--- sleuthkit-1.69/src/fstools/fs_dent.c    2004-01-06 17:50:52.000000000 -0500
+++ sleuthkit-1.69-frss/src/fstools/fs_dent.c       2004-05-23 20:34:15.000000000 -0400
@@ -54,9 +54,13 @@
 #include "mymalloc.h"
 #include "error.h"
 #include "ntfs.h"
+#ifndef __FRSS_H__
+       #include "frss.h"
+#endif

 extern char *tzname[2];

+
 /* ascii version of file types based on types given in fs_dent.h */
 char fs_dent_str[FS_DENT_MAX_STR][2] = {"-","f","c","","d","","b","","r","",
   "l","","s","h","w"};
@@ -295,9 +299,23 @@
   FS_DATA *fs_data)
 {
       FS_INODE *fs_inode = fs_dent->fsi;
-
       fs_dent_print(hFile, fs_dent, flags, fs, fs_data);
-
+/*FRSS INSERT START*/
+if (ent_report == 1){
+                       if (fs_dent->inode != 0){
+
+                       flags = FS_FLAG_CONT | FS_FLAG_ALLOC | FS_FLAG_UNALLOC;
+                       fs->file_walk(fs, fs_inode, 0,0, flags, frss_inode_scoring, "");
+                       ent_score = ent_calc();
+                       ascii_score = ascii_calc();
+                       fprintf(hFile, "\t%f\t%f",ent_score,ascii_score);
+                       clear_vars();
+                       }
+                       else{
+                                       fprintf(hFile,"\tNULL\tNULL");
+                       }
+       }
+/* FRSS INSERT END */
       if ((fs == NULL) || (fs_inode == NULL)) {

               fprintf(hFile, "\t0000.00.00 00:00:00 (GMT)");
@@ -439,4 +457,3 @@
       fprintf(hFile, "%lu|0\n", (fs) ? (ULONG)fs->file_bsize : 0);

 }
-
diff -aurN sleuthkit-1.69/src/fstools/fs_tools.h sleuthkit-1.69-
frss/src/fstools/fs_tools.h
--- sleuthkit-1.69/src/fstools/fs_tools.h   2004-04-18 15:49:42.000000000 -0400
+++ sleuthkit-1.69-frss/src/fstools/fs_tools.h      2004-05-20 14:08:56.000000000 -0400
@@ -44,7 +44,7 @@
   * Verbose logging.
   */
 extern FILE *logfp;
-
+int ent_report;
  /*
   * Solaris 2.x. Build for large files when dealing with filesystems > 2GB.
   * With the 32-bit file model, needs pread() to access filesystems > 2GB.
diff -aurN sleuthkit-1.69/src/fstools/icat.c sleuthkit-1.69-frss/src/fstools/icat.c
--- sleuthkit-1.69/src/fstools/icat.c       2004-01-06 17:50:52.000000000 -0500
+++ sleuthkit-1.69-frss/src/fstools/icat.c  2004-05-23 13:45:20.000000000 -0400
```

```
@@ -183,7 +183,6 @@
                        if (*cp || cp == dash)
                                usage();
                }
-
                inode = fs->inode_lookup(fs, inum);
                if (!inode)
                        error ("error getting inode");
diff -aurN sleuthkit-1.69/src/fstools/istat.c sleuthkit-1.69-frss/src/fstools/istat.c
--- sleuthkit-1.69/src/fstools/istat.c      2004-01-06 17:50:52.000000000 -0500
+++ sleuthkit-1.69-frss/src/fstools/istat.c 2004-05-23 20:41:53.000000000 -0400
@@ -53,7 +53,9 @@
 #include "fs_tools.h"
 #include "error.h"
 #include <time.h>
-
+#ifndef __FRSS_H__
+       #include "frss.h"
+#endif
 FILE   *logfp;


@@ -91,7 +93,8 @@
        printf("\t-b num: force the display of NUM address of block pointers\n");
        printf("\t-z zone: time zone of original machine (i.e. EST5EDT or GMT)\n");
        printf("\t-s seconds: Time skew of original machine (in seconds)\n");
-    printf("\t-f fstype: Image file system type\n");
+       printf("\t-x FRSS: Shows FRSS Scores\n");
+       printf("\t-f fstype: Image file system type\n");
        printf("Supported file system types:\n");
        fs_print_types();
        exit(1);
@@ -106,14 +109,14 @@
     char   *fstype = DEF_FSTYPE;
        FS_INFO        *fs;
        int32_t sec_skew = 0;
-
+       FS_INODE *inode;
        /* When > 0 this is the number of blocks to print, used for -b arg */
        int numblock = 0;

     progname = argv[0];


-    while ((ch = getopt(argc, argv, "b:f:s:vVz:")) > 0) {
+    while ((ch = getopt(argc, argv, "b:f:s:xvVz:")) > 0) {
        switch (ch) {
        default:
            usage();
@@ -127,6 +130,9 @@
        case 'f':
            fstype = optarg;
            break;
+       case 'x':
+           ent_report =1;
+             break;
        case 's':
                sec_skew = atoi(optarg);
                break;
@@ -152,9 +158,9 @@
        }
     }

-    if ((optind+2) != argc)
+    if ((optind+2) != argc){
                usage();
-
+}
```

```
        /*
       * Open the file system.
       */
@@ -172,9 +178,19 @@
                 (ULONG)fs->first_inum);
                return 1;
        }
-
+       int type =0;
+       int id = 0;
+       int     flags = FS_FLAG_CONT | FS_FLAG_ALLOC | FS_FLAG_UNALLOC;
+
+       inode = fs->inode_lookup(fs, inum);
        fs->istat(fs, stdout, inum, numblock, sec_skew);
-
-    fs->close(fs);
+       /*FRSS INSERT START*/
+       if (ent_report == 1){
+       fs->file_walk(fs, inode, type, id, flags, frss_sector_scoring, "");
+       printf("\nEntropy Score:%f\n",ent_score);
+       printf("Ascii Score:%f\n",ascii_score);
+       /* FRSS INSERT END */
+       }
+fs->close(fs);
      exit(0);
 }
diff -aurN sleuthkit-1.69/src/fstools/Makefile sleuthkit-1.69-frss/src/fstools/Makefile
--- sleuthkit-1.69/src/fstools/Makefile     2004-04-16 11:51:04.000000000 -0400
+++ sleuthkit-1.69-frss/src/fstools/Makefile 2004-05-19 19:51:31.000000000 -0400
@@ -6,11 +6,11 @@
 DEBUG  = -g
 INCL   = -I../misc
 CFLAGS = $(DEFS) $(INCL) $(OPT) $(DEBUG)
-LIBOBJ = fs_buf.o fs_inode.o fs_io.o fs_open.o \
+LIBOBJ = frss.o fs_buf.o fs_inode.o fs_io.o fs_open.o \
        fs_dent.o fs_types.o fs_data.o mylseek.o get.o \
        ffs.o ffs_dent.o ext2fs.o ext2fs_dent.o \
        fatfs.o fatfs_dent.o ntfs.o ntfs_dent.o swapfs.o rawfs.o
-LIBS   = ../misc/aux_lib.a
+LIBS   = ../misc/aux_lib.a -lm
 LIB    = fs_lib.a
 BIN_DIR       = ../../bin
 PROGS  = $(BIN_DIR)/ils $(BIN_DIR)/dls $(BIN_DIR)/icat \
```