A New Bioinformatics-Inspired and Binary Analysis: Coding Style/Motif Identification

Scott Miller
Offensive Computing
For that B guy
After drawing an analog from computer binary analysis to a similar analysis problem in Biology, a useful technique modeled after BLAST was developed. This technique has been implemented with the binBLAST tool suite, a collection of command line tools with an optional web interface. This tool suite allows consideration of binary executables on a per-instruction level that does not require executables to follow any coding or interface standards. Of particular interest, this technique seems sensitive to coding and interface standards and creates the potential for automatic code/style identification.
Acknowledgements

- NSF/Scholarship for Service
- NMIMT/CS Department
- Thesis committee
- Technical and general reviewers
- Offensive Computing
Overview

- Concept
- binBLAST program suite
- Application: Version analysis
- Work-in-progress
Motivation
System Defense

Current defense: policy, software, hardware

- New threats necessarily require patches
- Patches require verification
- Time available for verification is decreasing

The very tools we use to defend ourselves are contributing to the problem.
Bioinformatics Analogy

*Functional analysis from code is not new...*
Technique
Alignment Scoring Example

- Same concept as BLAST
- Scored string matching
  - Match adds one (+1)
  - Difference subtracts one (-1)
  - Actual scoring constants determined using Karlin-Altschul analysis

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>H</th>
<th>E</th>
<th>S</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Align #1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Align #2</td>
<td>T</td>
<td>H</td>
<td>E</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>Score</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Technique
Scoring for Binaries

- Modified BLAST approach
- Karlin-Altschul analysis on over 553,901,654 instructions
- Instructions reduced to 4-byte format

```
mov ebp, esp
```

- Operator class, truncated to one byte
- Operands, truncated to two bytes
- Operator, truncated to one byte
Overview

- Concept
- binBLAST program suite
- Application: Version analysis
- Work-in-progress
binBLAST Suite

- Major analysis tools
  - mklib.py – builds disassembly library/index
  - bincompare – comparison engine
  - filterbincompare.py – output filter to clean/to refactor
  - matchoutput.py – visualization tools

- Auxiliary tools
  - binblast_html.cgi – web-based interface
  - objdumputil.py – debugger interface
binBLAST Suite

mklib.py

$ ./mklib.py --help
usage: mklib.py [options]

options:
-h, --help                     show this help message and exit
-i FILE, --iso=FILE            Use an ISO file as the distribution source
-d DIR, --dir=DIR              Use a directory tree as the distribution source
-n MY_LINUX_X.X, --name=MY_LINUX_X.X
                                   A unique identifier for this distribution
-b, --bin                      Produce binary output, name.dat and name.idx
-v, --verbose                  Produce verbose status information
-o, --no-db                    Do not produce the default database output, name.db
$ ./bincompare --help
Options:
- h Print this help
- t=N: Set the threshold parameter to N, default is 10
- x1=N: Set scoring parameter x1 to N, default is 6
- x2=N: Set scoring parameter x2 to N, default is 5
- x3=N: Set scoring parameter x3 to N, default is 4
- x4=N: Set scoring parameter x4 to N, default is -4
Files:
name - The name of the comparison file, i.e. name.dat
offset - The start offset in quadwords
        Relative to the start of the file
        Defaults to 0
len - The length for comparison in quadwords
        Defaults to the name's byte length / 4
The fileA length must be smaller (or the same size) as fileB length.
$ ./filterbincompare.py --help
usage: filterbincompare.py [options]

options:
  -h, --help                show this help message and exit
  -d, --no-dirout           Do not correct dirout relative offsets to absolute offsets
  -n, --nosort              Don't sort results (reduces memory usage)
  -t MINSCORE, --threshold=MINSCORE
                            Threshold minimum score, if score is above Karlin-Altschul threshold
  -l MINLEN, --minlen=MINLEN
                            Minimum match length
  -a, --filea               Reduce results into matches covering only fileA
binBLAST
matchoutput.py

$ ./matchoutput.py --help
usage: matchoutput.py [options]

options:
  -h, --help       show this help message and exit
  -s, --similarity Create a similarity table (LaTeX format)
  -d, --distance   Create a distance table (PHYLIP format)
  -m, --motifs     Create motif groups in graph
  -g, --graph      Create a graph of the matches
                   (graphviz dot format)
  -a, --assembly   Print aligned assembly (requires IDXFILEA STARTA, LENA
                   IDXFILEB STARTB, LENB as arguments)
binBLAST Suite
Process Overview/Demo

- Add programs using mklib.py
- Perform comparison using bincompare
- Pipe output of bincompare through filterbincompare.py
- Pipe output of filterbincompare.py through matchoutput.py

*Except for mklib.py, tools use stdin/stdout for data passing*
binBLAST Web Interface

/ usr/ bin/ file. exe - / var/ www/ workdir/ cygwin. tgz - / var/ www/ workdir/ uploads

Match coverage: 0.963743

Add a file - Run a query - View results

binBLAST - Display query - knoppix-finger-cygwin-file.out - Matches - Mozilla Firefox

FileA FileB
Score Len Name Offset Name Offset
105 29 /usr/bin/file.exe 741 /usr/bin/finger 1123
104 27 /usr/bin/file.exe 354 /usr/bin/finger 2867
102 47 /usr/bin/file.exe 736 /usr/bin/finger 1127
99 45 /usr/bin/file.exe 739 /usr/bin/finger 1127
99 36 /usr/bin/file.exe 739 /usr/bin/finger 1116
99 26 /usr/bin/file.exe 356 /usr/bin/finger 814
98 41 /usr/bin/file.exe 724 /usr/bin/finger 1866
96 48 /usr/bin/file.exe 736 /usr/bin/finger 1117
95 28 /usr/bin/file.exe 746 /usr/bin/finger 1875
Overview

- Concept
- binBLAST program suite
- **Application: Version analysis**
- Work-in-progress
## Application: Version Analysis

### Sendmail

<table>
<thead>
<tr>
<th>Program</th>
<th>Distribution</th>
<th>Date released</th>
<th>Likely compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>postfix-2.2.8</td>
<td>SUSE (i386)</td>
<td>1/7/2006</td>
<td>gcc 3.4.5</td>
</tr>
<tr>
<td>chromium-0.9.12</td>
<td>SUSE (i386)</td>
<td>9/13/2005</td>
<td>gcc 3.4.4</td>
</tr>
<tr>
<td>sendmail-8.13.5</td>
<td>SUSE (i386)</td>
<td>9/16/2005</td>
<td>gcc 3.4.4</td>
</tr>
<tr>
<td>sendmail-8.13.4</td>
<td>Fedora (i386)</td>
<td>3/27/2005</td>
<td>gcc 3.3.6</td>
</tr>
<tr>
<td>sendmail-8.12.9</td>
<td>Mandrake (i586)</td>
<td>3/29/2003</td>
<td>gcc 3.2.2</td>
</tr>
<tr>
<td>sendmail-8.9.1</td>
<td>Redhat (i386)</td>
<td>7/7/1998</td>
<td>gcc 2.8.1</td>
</tr>
</tbody>
</table>
## Application: Version Analysis

### Raw Results

<table>
<thead>
<tr>
<th>Score</th>
<th>Len</th>
<th>File A</th>
<th>File B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Name</td>
<td>Section</td>
</tr>
<tr>
<td>5095</td>
<td>899</td>
<td>sendmail-8.13.5</td>
<td>_init</td>
</tr>
<tr>
<td>4829</td>
<td>905</td>
<td>sendmail-8.12.9</td>
<td>_init</td>
</tr>
<tr>
<td>4797</td>
<td>899</td>
<td>sendmail-8.12.9</td>
<td>_init</td>
</tr>
<tr>
<td>4588</td>
<td>931</td>
<td>sendmail-8.13.5</td>
<td>_init</td>
</tr>
<tr>
<td>4488</td>
<td>907</td>
<td>sendmail-8.12.9</td>
<td>(_init)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

*Top results are associated with GCC-3 automatic code templates*
Code templates common in GCC-3

Typical GCC-3 entry point

GCC-3 external symbol resolutions

Most common GCC-3 external call invocation
Application: Version Analysis
Phylogenetic Analysis

- PHYLIP Unweighted Pair Group Method with Arithmetic mean (UPGMA)
- Sendmail 8.9.1 distant because of use of GCC-2 compiler instead of GCC-3
Both the UPGMA (left) and Nearest-Neighbor (right) methods generate similar phylograms that reflect known relationships.
Review

- BLAST technique can be used on binaries
- binBLAST: an open-source analysis suite
- Sensitive to compiler code generation
- Detected relationships that mirror known relationships
Overview

- Concept
- binBLAST program suite
- Application: Version analysis
- Work-in-progress
Work in Progress

- Improved web interface
- Automatic malware classification
- Automatic antivirus signature generation
- Motif identification/sub-alignment match removal
- Wiki-type collaborative RE environment

Suite available at
http://www.offensivecomputing.net
Selected References

- **BLAST**: S. Altschul et al., *Basic Local Alignment Search Tool*, 1990
- Malware: *Offensive Computing*, 2006
- Structural analysis: H. Flake, *Structural Comparison of Executable Objects*, 2004