Rendezvous: A search engine for binary code

Wei Ming Khoo, Alan Mycroft, Ross Anderson
University of Cambridge

MSR 2013
19 May 2013

Demo: http://www.rendezvousalpha.com
To audit or not to audit

You can’t trust code that you did not totally create yourself (Ken Thompson, 1984)

• Engineering: Software quality
  - ‘CVE top 20’, bugtraq, App Store “Bouncers”
  - Diebold voting machines’ crypto (e.g. [Yasinsac’07])

• Legal: Software compliance
  - EU data protection directive 95/46/EC (2012)

• Legal: Software 3rd-party licensing
  - GPL non-compliance: Apple (GNU Go in Appstore 2010) and Microsoft (Win7 USB/DVD download tool 2009) included
Software reverse engg.

Software RE is sometimes necessary for audit

- Source code not always available
  - Third-party sub-contractors, sub-sub-contractors, app store publishers
- “What you see [in the source] is not what you execute” [Balakrishnan, Reps 2005]

- Decompilers
  - Boomerang, REC Studio 4, Anatomizer, Andromeda, exetoc, desquirr
  - Current state-of-the-art: Hex-Rays, US$1,160 per license per year + expertise
  - 415 man-hours to decompile 1,500 LoC comprising 8% of code base [VanEmmerik’04]
But, code reuse is prevalent

And increasingly so due to advances in software mining and SBSE

• Catalysts include market competitiveness, application complexity, quality of reusable components [Schmidt’99, ’00, ’06]

• Six open source projects: On average 74% of code base was external [Haefliger’08]

• Sometimes illegally: >250 products found GPL non-compliant, most famously Linksys WRT54G
Proposed solution

Search-based reverse engineering (SBRE)

“How do we decompile?” with “Given a candidate decompilation, how good a match is it?”

Same shift occurred for statistical machine translation
Take away slide

• Software RE is tedious but sometimes necessary for audit

• Code reuse is common in software

• We propose reframing: **software RE as a search problem**, relying on existing software to obtain source code

• Q: How can we do this in a way that is compiler-agnostic?
How we achieve this

• Design trade-offs
• Feature extraction
• Indexing & Querying
• Experimental results
Design space

• We want features that can uniquely identify functions
• We want speed + accuracy: We chose **Speed** first
• Speed meant that we chose static over dynamic analysis (Assumption: no obfuscation)
• We studied heuristic features from existing literature that can be extracted directly from a disassembly:
  - Instruction mnemonics
  - Control-flow sub-graphs
  - Data constants
Feature extraction

Disassemble

Tokenise

Token-specific processing

Executable

Disassembly

Mnemonic n-grams

Control-flow sub-graphs

Data Constants

Alphabetic strings (Query terms)
Instruction mnemonics

• Instruction mnemonic (textual) differs from an opcode (hex), e.g. \(0x8b\) (load) and \(0x89\) (store) map to ‘\textit{mov}’

• Assume a Markov property, \(n^{th}\) token is influenced by the previous \(n - 1\) tokens

• Considered \(n = 1, 2, 3, 4\)

\(\text{push, mov, push} \rightarrow 0x73f973 \rightarrow \text{XvxFGF}\)
Control-flow $k$-graphs

- $k$-graph is a connected sub-graph comprising $k$ nodes, compute them all ($k = 3, 4, 5, 6, 7$)
- Convert to $k$-by-$k$ matrix and compute its canonical form, rep as string (Nauty graph library)
Constants

• Empirical observation that data constants do not change with compiler or options

• Considered 32-bit integers and strings

• Immediate operands, pointer offsets (excluding stack and frame pointer offsets)

• Integer may be an address, do a lookup
Indexing & querying
## Results at a glance

<table>
<thead>
<tr>
<th>Model</th>
<th>glibc $F_2$</th>
<th>coreutils $F_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best $n$-gram (4-gram)</td>
<td>0.764</td>
<td>0.665</td>
</tr>
<tr>
<td>Best $k$-graph (5-graph)</td>
<td>0.706</td>
<td>0.627</td>
</tr>
<tr>
<td>Constants</td>
<td>0.681</td>
<td>0.772</td>
</tr>
<tr>
<td>Best mixed $n$-gram (1+4-gram)</td>
<td>0.777</td>
<td>0.671</td>
</tr>
<tr>
<td>Best mixed $k$-graph (5+7-graph)</td>
<td>0.768</td>
<td>0.657</td>
</tr>
<tr>
<td>Best composite (4-gram/5-graph/constants)</td>
<td>0.867</td>
<td>0.830</td>
</tr>
</tbody>
</table>

**Precision**

$$\text{precision} = \frac{tp}{tp + fp}$$

**Recall**

$$\text{recall} = \frac{tp}{tp + fn}$$

$$F_2 = \frac{5 \cdot (\text{precision} \cdot \text{recall})}{4 \cdot \text{precision} + \text{recall}}$$

Combining features increases $F_2$, implying independence.
Conclusion

• Software RE is tedious (but sometimes necessary) for audit
• Code reuse is common in software
• We propose reframing: software RE as a search problem
• Able to achieve $F_2$ rates of 0.867 & 0.830 combining mnemonics, $k$-graphs and constants

http://www.rendezvousalpha.com