“You’re KIL-ling me” – Implementing Knowledge Interaction Levels in a Cyber Analysis tool

Adam R. Bryant, Ph.D.

Cyber Research Laboratory, Riverside Research, Beavercreek, OH, USA
Introduction

• Enlisted missile maintainer (2M0)
• Comm-Info Officer (33S)
• Civil service researcher
• Not-for-profit R&D
Researching How To Make Better Binary Analysis Tools

• Reverse engineering malware (or other executable code) is challenging
  • Little tool support for abstractions
  • Proceduralized / tacit knowledge
  • Problem-solving strategies vs. reading bytes

• Want to make things easier
  • Better abstractions
  • Support for working memory and attention requirements
Characteristics of Reverse Engineering Problems

• More data than you can reasonably process
• High-level implications of low-level relationships
• Underlying structural, functional, and behavioral relationships not necessarily present in the data
• Reasoning tasks rely on background knowledge
• Data exists at multiple layers of abstraction
Why Reverse Engineering?

• Practical cyber security implications
• Figure out what malware does
• Find hidden vulnerabilities / uncover faulty assumptions

• Things real people reverse engineer:
  • Applications, operating systems, networks, protocols
  • Weapon systems, PCS / SCADA systems
  • Devices, processors, game consoles, tablets, cell phones
Making it Easy for Non-Reverse Engineers

• It’s important, but it needs to be easier
• Especially for the people that need it:
  • New “cyber” operators in the military
  • Network administrators (≠ reverse engineers)
  • Folks that make decisions
• Hands-on training in cyber is a huge need
  • Quickly acquire expert mental models
  • Increase decision-making capability
Embedded Training

• Embed the training in the tool
  • Design good abstractions for complex tasks
  • Structured / integrated way to “bring people along”
  • Let them experience success along the way
  • Enough challenge to keep them interested
  • Keep them from getting lost

• Some folks are really good at this:
Mental Models of Programs

• What do we know about people’s mental models of executable programs?
• Focused around some key abstractions:
  • Functions
  • Modules
  • Basic Blocks
  • Instructions
  • Data flow
• Low-level details → high level abstractions?
• Generating hypotheses
• Seeking information
What are People Doing When They Look at Binary Code?

- How do we model the transformation of low-level implementation up to concepts (and then down again?)

Conceptual Model of Reverse Engineering from (Byrne, 2002)
How Do We Do Some of That Automatically?

- How can we discover and locate these concepts?
- What concepts should we be looking for in the code?
- How do we match concepts to their locations in the code?
Concepts in Reverse Engineering

- Comprehension relies on:
  - Intensions (understanding concepts)
  - Extensions (locating/tracing concepts in the code)
  - Naming the concepts

“Concept Triangle” Model of Program Comprehension (Rajlich, 2009)
Causal Relationships

- Relationships between abstract concepts and data can be specified (and measured)
- What are the causes and effects we care about in reverse engineering?

![Diagram of causal relationships]

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Generalizing data to concepts</td>
</tr>
<tr>
<td>2</td>
<td>Matching concepts to data</td>
</tr>
<tr>
<td>3</td>
<td>Showing causal relations in data</td>
</tr>
<tr>
<td>4</td>
<td>Generalizing causal relations in concepts</td>
</tr>
<tr>
<td>5</td>
<td>Refining quality of concepts</td>
</tr>
</tbody>
</table>
Zhang, et al. (2009)

The iterations proceed from exploratory to focused search and sense-making.

- Task/Problem
- Existing Knowledge
- Structures and their instantiations with data
- Identification of Gaps
- Decision/Solution/Task completion
- Data loop
- Search: exploratory/focused
- Searching for data
- Instantiating structure
- Building structure
- Structure loop
- Structure gap
- Data gap
- Outcomes
- Updated knowledge
- Accretion: Instantiated structure
- Tuning: Adapted structure
- Re-structuring: New structure

Engineering Challenges

- Interfacing with the system
- How to present these abstractions easily
- Where do people get tripped up?
- How to determine someone’s goal from actions?
- How to present tasks
- Measuring learning
Designing a Semi-Autonomous System

• Decomposing:
  • Knowledge
  • Tasks and actions
  • Information needs
Decomposing is Modeling

- Modeling semantic knowledge
  - OWL and RDF triples
- Modeling procedural knowledge
  - Rules and pattern-matching
  - State diagrams
- Modeling tasks
  - Record sessions ("click, click, scroll, click, scroll, click")
  - Annotate action sequences (action A, B, C, …)
  - Organize sequences into goals

**concept** structure = declarative knowledge

**process** structure = procedural knowledge
Decomposing is Modeling

• Modeling the system
• Annotating UI elements
• Capturing system “states”
• Tie semantic data to sequences of actions

• Reassemble:
  • Action sequences to unit tasks
  • Unit tasks to goal-driven Tasks
  • Goals back to top-level Goals
Modeling System as Components

Example:

**Basic Block 1**
Address 4015F0  
Prior-to BasicBlock2  
CanSelect true  
CanHover true  
Constraint  
If EBX < 5

**Basic Block 2**
Address 40CFFC  
Label  
CanModify true  
CanHover true  
Captures User Input
Knowledge-Enabled Strategies

• Knowledge separates exploratory problem-solving strategies from exploitation strategies
• Identify where barriers exist
Where We’re At Today?

- Agent Layer
- State Layer (agent)
- State Layer (system)
- Interface Layer
- System Layer

- Long-term memory
- Working memory
- Rules
- Mental model

- Goals
- Situation
- percepts
- actions
- events
- behaviors
- affordances
- information displays
- UI objects
- data structures
- subroutines
- modules
- bytes
- packets
The System and Interface Layer: Our DigR Reverse Engineering Framework

- Stealthy x86 disassembler
  - Integrates recursive descent and linear sweep analysis

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Section</th>
<th>Label</th>
<th>Bytes</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>004124A3</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>8b 0c bd 54 5b...</td>
<td>mov ecx, [edi*4+415B54h]</td>
</tr>
<tr>
<td>004124AA</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>50</td>
<td>push eax</td>
</tr>
<tr>
<td>004124AB</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>53</td>
<td>push ebx</td>
</tr>
<tr>
<td>004124AC</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>51</td>
<td>push ecx</td>
</tr>
<tr>
<td>004124AD</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>57</td>
<td>push edi</td>
</tr>
<tr>
<td>004124AE</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>e8 7d 02 00 00</td>
<td>call sub_412730</td>
</tr>
<tr>
<td>004124B3</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>83 c4 10</td>
<td>add esp, 10h</td>
</tr>
<tr>
<td>004124B6</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>85 c0</td>
<td>test eax, eax</td>
</tr>
<tr>
<td>004124B8</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>0f 85 22 01 00...</td>
<td>jnz loc_4125E0</td>
</tr>
<tr>
<td>004124BE</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>32 c0</td>
<td>xor al, al</td>
</tr>
<tr>
<td>004124C0</td>
<td>.text:sub_4123E0</td>
<td></td>
<td>eb 02</td>
<td>jmp short loc_4124C4</td>
</tr>
</tbody>
</table>
```
Breakpoint Management

• Both stealthy and persistent breakpoints
  • Stealthy breakpoints use re-routing tricks to circumvent malware anti-analysis routines
Graph-Based Workflow / Navigation

• Adding features to abstract binary navigation tasks
  • Meaningful icons
  • Maps
  • Index-search
Integrated Binary Editing

• Makes it easy to change functionality and export to a new binary
Semantic Sub-routine Annotation

- Allows you to tag subroutines with higher-level information
- Lets you then use that information to search for locations
Function Trace Profiling

- Records information from dynamic trace of execution
Integrated Static / Dynamic Analysis

- Integrates static and dynamic execution information to help find “interesting” areas
Blackboard AI System

• Processes action streams from:
  • Users and UI actions
  • The binary and its state configuration changes

• Currently defining higher-level rules for RE tasks
What’s Next?

• Agent knowledge models for specific tasks
• State-layer abstractions for the agent and the task
• Break up knowledge into separate interaction levels
• Maybe make some tools to make this easier?
Take Aways

• Can embed training in a system, but it requires careful knowledge design
• That same process can lead to decision-making support for users
• This is a growing field, but we need better tools to support knowledge-level modeling … it’s challenging, but doesn’t have to be AS difficult as it is
Contact Info:

Adam Bryant, Ph.D.
Cyber Research Laboratory
Riverside Research
abryant@riversideresearch.org
(937) 427-7080