Binary Analysis: Part 10

Ryan Goluch

Spring 2020

Where We Left Off…

- Initial testing of:
  - Radare
  - Disassembled Binary loaded into Atlas
  - CFG Generation
- Planning on implementing:
  - Path counting
  - Loop, Branch, and Exit node counting
  - Comparing binary counts to source counts

Where We Are…

- Completed:
  - Path Counting
  - Loop, Branch, and Function Exit Node Counting
  - Comparison File Generation
  - Link to code repository
- Loop, Branch, and Function Exit Node counting updates CFGs with appropriate XCSG tags
- What would still need to happen?
  - “Interpretation” or “Verification” Stage
    - Have all the numbers (they seem to be accurate)
    - Need to determine their meaning/implication

Xinu: By The Numbers

Function Path Count Comparison:

- Binary Paths > Source Paths: 18 Functions
- Binary Paths < Source Paths: 70 Functions
- Binary Paths == Source Paths: 46 Functions

Other Notes:

- Binary Functions with 1 Node: 56
- Binary Functions with 1 Node + Self Loop: 23

Total Function Count: 213

Function Highlights: Path Counts

- `test_memory`
  - 60 paths found in source
  - 0 paths found in binary
- `test_libStdio`
  - 1 path found for source
  - 4,294,967,296 paths found for binary
  - Possible Cause: Something to do with it being a test file and the way that it’s written or use of things like `failif`
  - Possible Cause: Could be due to how it is compiled and disassembled from radare
- `test_libLimits`
  - 32,768 paths found in source
  - 1 path found in binary

Function Highlights: Path Counts

- `convertDate`
  - 180 paths found in source
  - 2576 paths found in binary
- `xsh_memstat`
  - 483 paths found in source
  - 1,347 paths found in binary
- `test_mailbox`
  - 1 path in source
  - 207,360 paths in binary
- `test_libString`
  - 223,948,800 paths in source
  - 149,299,200 paths in binary

Path Count Mismatch: Possible Causes

- Short circuiting
  - Nested `if` statements
- Optimization
  - Variables being shared by conditional statements
- The way test files are written in source vs what they become in binary
- Disassembled code only being an “intermediate” state
  - Not full source code

Xinu: By The Numbers Pt. 2

- Loops
  - Binary > Source: 22 Functions
  - Binary < Source: 21 Functions
  - Binary == Source: 180 Functions
- Branches
  - Binary > Source: 84 Functions
  - Binary < Source: 20 Function
  - Binary == Source: 119 Functions
- Function Exit Nodes
  - Binary > Source: 24 Functions
  - Binary < Source: 90 Functions
  - Binary == Source: 109 Functions

Loop, Conditional, & Exit Observations

- Results seem to be reasonable
  - What you would expect to find given how code and compilation work
- Example: Binary Branch Count > Source Branch Count
  - Reasoning: Short circuits and optimizations
- Struggled with finding correctly writing the logic to identify loops
  - Needing to figure out the best possible use of tagging
  - Initially had forgotten to consider “Do...While” loops in source

Answering Previous Questions

- Does the control structure change from the source code to the binary?
  - Answer: Yes. I think you can say that compilation does in fact modify the control structure due to things such as optimization and short circuiting. We can also see this in path counts for some of the simpler functions.
- Can you verify a piece of source code based on the disassembled binary?
  - Answer: TBD. I think that there would need to be more investigation done into the results this tool produces as well as establish a definition of “validate” in order to definitively answer this question.

Key Takeaways

- Path count for some functions
  - In the hundreds of thousands or millions
  - Possible unit to measure complexity?
  - Does this provide a better metric than LOC?
  - Can we say for sure that large LOCs => Higher Complexity?
- Interesting problems to reason about
  - How to identify and correctly tag parts of CFGs
  - Example: What does it really mean for something to be a loop in a CFG?

Key Takeaways

- Application of Formal Methods
    - Seemed to apply the thought process or reasoning developed in Formal Methods
    - Not necessarily directly applying content, but ideas
- Interested to see where this could go

Remaining Questions

- Is it possible to do data flow analysis?
  - Is it worth it to do data flow analysis?
- How can we address the self loop functions?
- Can this be used with stripped or packed binaries?
- What are the implications of these results or what insights do they provide?
- How can these techniques and tools be built on to aid in security research?
  - Malware analysis, RE, etc.

Thank you!