Post-Intrusion Memory Forensics Analysis

Pengfei Sun, Rui Han†, Saman Zonouz
Dept. of Electrical and Computer Engineering, Rutgers University, NJ
† Dept. of Electrical and Computer Engineering, University of Miami, FL

Introduction and Motivation

Memory Dump Forensics Analysis

- Accurate memory dump layout
  - Stripped binary no source code
  - Without execution traces
- Use cases: Any of the running processes within the system

Current approaches

- Binary executable analysis techniques extract data structures defined within an executable by disassembly or symbolic execution
- Dynamic execution analysis solutions which trace the execution to reverse engineer data types using type-revealing instructions
- Static memory analysis perform forensics directly on memory dumps

Limitations

- Limited use for memory forensics when the execution trace is not available
- Dynamic execution monitors cause a very high performance overhead
- Static memory analyses are not sufficiently accurate in practice

Our Approach and Contribution

Approach

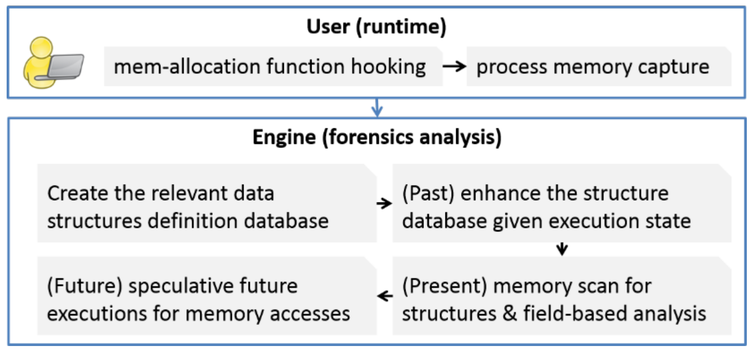
We present a hybrid memory forensics solution that leverages the high accuracy of static executable analyses *(i)* and dynamic execution monitoring *(ii)* to provide a low overhead and precise static memory dump forensics *(iii)* when the execution trace is not available.

Contribution

- We introduce a trace-free memory data structure forensics solution with high reverse engineering accuracy and negligible 1.8% runtime overhead.
- We present a probabilistic information fusion method to combine prior statistical information about possible past traces, results from the present dump forensics and speculative investigation of potential future executions of the suspended process.
- We present a preliminary evaluation on real-world settings, i.e., CoreUtils suit.

Our Approach: Architecture

The High-Level Architecture



Present

- The engine performs a forensics analysis of the captured dump using the created models (data structure definition database and statistical information).
- It sweeps the dump for landmark signatures (inserted by the hooked API), and extracts every structure's base address and size.
- The engine marks every memory address with possible data types using the memory value and the engine's forensics rules.
- The engine uses its forensics results and the created models to calculates a ranked list of the best matching data structures for each memory location.

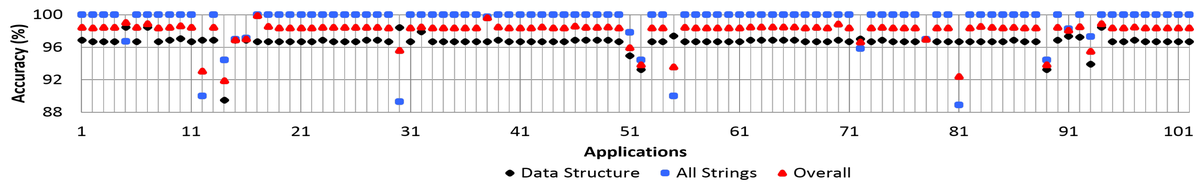
Past

- The engine automatically executes static code analysis to investigate available library and kernel sources searching for data structure definitions with the context.
- The engine explores and analyzes the memory dump's possible past execution trace. It starts from the executable's entry point, and statically explores the call and control flow graphs for possible paths to the captured dump's execution state.
- Using directed symbolic execution, the engine filters out the infeasible paths and generates the corresponding test cases. Through an instrumented execution of the test cases, the engine logs the structure memory allocations and collects statistical information about the potential structures on the captured dump.

Future

- The engine revives the captured dump's execution, explores all feasible future branches.
- The engine reverse engineers data type revealing instructions and the library/system calls with known return/argument data types.
- The engine implements backward data taint analysis to backtrack the revealed data types to a memory address of the captured dump.

Evaluation



We measured the accuracy of our solution's ultimate data type forensics. The above figure shows the results for strings and other data structures of each application separately. Our solution correctly recognized 99% of the strings and 96.7% of the memory dump data structures correctly. Its overall accuracy level 98.1% is very promising even though our solution did not have access to the execution traces before the memory capture point.

Acknowledgements

We appreciate the anonymous reviewers and thank our sponsor, the Office of Naval Research (Grant N00014-15-1-2741), for their support and constructive continuous feedback on this project.