What is Data Structure Reverse Engineering

Data structure reverse engineering aims to infer both the
- Syntax (layout, size, offset) of variables
- Semantics (meaningful use, e.g., ip_addr, pid_t) of Types
  when only given a binary code.

Who cares

- Memory Forensics
- Vulnerability Discovery
- Protocol Reverse Engineering
- Program signature
- Virtual machine introspection
- ...

Understanding the Compilation

Credit: figure is from Jonghyup Lee’s NDSS’11 presentation
Our goal

Again, from binary code, infer

- **Syntax** (layout, size, offset) of variables
- **Semantics** (meaningful use, e.g., `ip_addr`, `pid_t`) of Types

Three papers

- **REWARDS**: Automatic Reverse Engineering of Data Structures from Binary Execution. NDSS 2010
- **TIE**: Principled Reverse Engineering of Types in Binary Programs. NDSS 2011
- **Howard**: A Dynamic Excavator for Reverse Engineering Data Structures. NDSS 2011

Outline

- Overview
- **REWARDS**
- **TIE**
- **HOWARD**
- Summary
### Key Ideas

- **Identifying** type resolution points in binary code
- **Tracking** data flow to resolve other all involved memory/register types

### Data Flow Tracking

**Standard technique:**
- Using shadow memory to keep the variable attributes and track the propagation

**New challenges:**
- Two-way type resolution
- Dynamic life time of stack and heap variables
- Memory locations will be reused
- Multiple instances of the same type

---

### Type Resolution points

- **System calls:** syscall num (eax)
  - **Syscall Enter:** Type parameter passing registers (i.e., ebx, ecx, edx, esi, edi, and ebp) if they involved
  - **Syscall Exit:** Type return value (eax)

- **Other type revealing instructions in binary code:**
  - String related (e.g., MOV/S/B/D/W, LOADS/STOS/B/D/W)
  - Floating-point related (e.g., FADD, FABS, FST)
  - Pointer-related (e.g., MOV (%edx), %ebx)

---

### Observation

- **Type Resolution points**

- **System calls:** syscall num (eax)
  - **Syscall Enter:** Type parameter passing registers (i.e., ebx, ecx, edx, esi, edi, and ebp) if they involved
  - **Syscall Exit:** Type return value (eax)

- **Other type revealing instructions in binary code:**
  - String related (e.g., MOV/S/B/D/W, LOADS/STOS/B/D/W)
  - Floating-point related (e.g., FADD, FABS, FST)
  - Pointer-related (e.g., MOV (%edx), %ebx)
Implementations

Dynamic Binary Instrumentation: A pin-tool on top of Pin-2.6
http://www.pintool.org/

Data structures of interest
- File information
- Network communication
- Process information
- Input-influenced (for vulnerability discovery)

Limitation

- Loss of data structure hierarchy
  - There is no corresponding type resolution point with the
    same hierarchical type
  - Heuristics exist (e.g., AutoFormat [Lin et al, NDSS2008])
- Path-sensitive memory reuse
  - Compiler might assign different local variables declared in
    different program paths to the same memory address
  - Path-sensitive analysis
- Type cast
  - int a=(float)b;

Conclusion

- REWARDS
  - Binary only
  - Dynamic analysis
  - Data flow tracking
- Key insight
  - Using system call/API/Type revealing instruction as type
    resolution point
  - Two-way type resolution
  - Unique benefits to memory forensics and vulnerability
    discovery

Goal I: Variable Discovery

Credit: figure is from Jonghyup Lee's NDSS'11 presentation
**Goal II: Type Inference**

Problems: multiple possible types

Credit: figure is from Jonghyup Lee’s NDSS’11 presentation

**TIE’s Contribution**

Key Idea

- Collecting Type Constraint from
  - Standard Library Call
  - System Call
  - Type Revealing Instructions
- Solving Type Constraint based on the Type Lattice
  - Equality ($A = B$), essentially Type Unification
  - Subtype relation ($A <: B$), using unification closure algorithm
  - Conjunctive ($A \land B$)
  - Disjunctive ($A \lor B$)

Limitations

- Works on regular programs compiled from C code
- Not very informative for irregular programs
- Infers types as what is in the TIE type system only

Conclusion

- Principled reverse engineering
- Well defined process + Type theory
- Type inference with more expressive type system (unification closure algorithm)
**Key Idea**

Observe how memory is used at runtime to detect data structures. For instance, if A is a pointer, then if
- A is a function frame pointer, then *(A + 8) is perhaps a function argument
- A is an address of a structure, then *(A + 8) is perhaps a field in this structure
- A is an address of an array, then *(A + 8) is perhaps an element of this array

**Intuition in HOWARD**

**Contributions**

- Increased coverage
  - Leverage S2E/Klee to increase the path coverage
- Can discover array and more internal data structure
  - Look for chains of accesses in a loop (e.g., elem = next++)
  - Look for sets of accesses with the same base in a linear space (e.g., elem = array[i];)
  - Look for chains of accesses in a loop
    - Boundary elements accessed outside the loop
    - Nested loops
    - Multiple loops in sequence
- Boundary elements accessed outside the loop
- Nested loops
- Multiple loops in sequence

**Summary**

All three techniques: from data use to infer syntax (offset, size, layout) of the variables

**REWARDS**

- REWARDS makes a first step in reconstructing the syntax and semantics of data structure, and demonstrated the security benefits of such reversing.

**TIE**

- TIE is able to statically analyze the binary code, and figure out the types for each individual variables

**HOWARD**

- HOWARD integrates symbolic execution to have a larger coverage than REWARDS, and also it focuses more on the internal data structures defined in the program.