Native x86 Decompilation Using Semantics-Preserving Structural Analysis and Iterative Control-Flow Structuring

Edward J. Schwartz*, JongHyup Lee†, Maverick Woo*, and David Brumley*

Carnegie Mellon University*
Korea National University of Transportation†
push %ebp
mov %esp,%ebp
sub $0x10,%esp
movl $0x1,-0x4(%ebp)
jmp 1d <f+0x1d>
mov -0x4(%ebp),%eax
imul 0x8(%ebp),%eax
mov %eax,-0x4(%ebp)
subl $0x1,0x8(%ebp)
cmpl $0x1,0x8(%ebp)
jg f <f+0xf>
mov -0x4(%ebp),%eax
leave
ret

Which would you rather analyze?

Functions

Variables

Control Flow

Types

int f(int c) {
    int accum = 1;
    for (; c > 1; c--) {
        accum = accum * c;
    }
    return accum;
}
```c
int f (int x) {
    int y = 1;
    while (x > y) {
        y++;
    }
    return y;
}
```

```c
int f (int a) {
    int v = 1;
    while (a > v++)
    {}
    return v;
}
```

**Compiled Binary**

```
010100101010101
001010110111010
101001010101010
101111100010100
010101101001010
100010010101101
010101011010111
```
Decompilers for Software Security

• **Manual reverse-engineering**
  – Traditional decompiler application

• **Apply wealth of existing source-code techniques to compiled programs** [Chang06]
  – Find bugs, vulnerabilities

• **Heard at Usenix Security 2013, during Dowsing for Overflows**
  – “We need source code to access the high-level control flow structure and types”
Desired Properties for Security

1. Effective abstraction recovery
   – Abstractions improve comprehension
Effective Abstraction Recovery

```
s1;
while (e1) {
    if (e2) { break; }
    s2;
}
s3;
```

More Abstract

```
s1;
L1: if (e1) { goto L2; }
    else { goto L4; }
L2: if (e2) { goto L4; }
L3: s2; goto L1;
L4: s3;
```

Less Abstract
Desired Properties for Security

1. Effective abstraction recovery
   – Abstractions improve comprehension

2. Correctness
   – Buggy(Decompiled) ➔ Buggy(Original)
Are these two programs semantically equivalent?
Prior Work on Decompilation

• Over 60 years of decompilation research

• Emphasis on manual reverse engineering
  – Readability metrics
    • Compression ratio: $1 - \frac{\text{LOC decompiled}}{\text{LOC assembly}}$
    • Smaller is better

• Little emphasis on other applications
  – Correctness is rarely explicitly tested
The Phoenix C Decompiler
How to build a better decompiler?

• Recover missing abstractions one at a time
  – **Semantics preserving** abstraction recovery
    • Rewrite program to use abstraction
    • Don’t change behavior of program
    • Similar to compiler optimization passes
Are these two programs semantically equivalent?
How to build a better decompiler?

• Recover missing abstractions one at a time
  – Semantics preserving abstraction recovery
    • Rewrite program to use abstraction
    • Don’t change behavior of program
    • Similar to compiler optimization passes

• Challenge: building semantics preserving recovery algorithms
  – This talk
    • Focus on control flow structuring
    • Empirical demonstration
Phoenix Overview

New in Phoenix

```c
int f (int x) {
    int y = 1;
    while (x > y) {
        y++;
    }
    return y;
}
```
Control Flow Graph Recovery

- Vertex represents straight-line binary code
- Edges represent possible control-flow transitions
- **Challenge**: Where does `jmp %eax` go?
- Phoenix uses Value Set Analysis [Balakrishnan10]
Type Inference on Executables (TIE) [Lee11]

How does each instruction constrain the types?

\texttt{movl} (%eax), %ebx

- Constraint 1: %eax is a pointer to type \texttt{<a>}
- Constraint 2: %ebx has type \texttt{<a>}
- Solve all constraints to find \texttt{<a>}

8/15/13 Usenix Security 2013
Control Flow Structuring
Control Flow Structuring

if (e) {
  ...
} else {
  ...

Compilation

Control Flow Structuring

if (e) {
  ...
} else {
  ...

if (e) {
  ...
} else {
  ...

8/15/13

Usenix Security 2013
Control Flow Structuring: Don’t Reinvent the Wheel

- Existing algorithms
  - Interval analysis [Allen70]
    - Identifies intervals or regions
  - Structural analysis [Sharir80]
    - Classifies regions into more specific types

- Both have been used in decompilers

- Phoenix based on structural analysis
Structural Analysis

• Iteratively match patterns to CFG
  – Collapse matching regions

• Returns a skeleton: while (e) { if (e’) {...} }

if-then-else

while
...;
while (...) { if (...) {...} else {...} };
...; ...;
Structural Analysis Property Checklist

1. Effective abstraction recovery
Structural Analysis Property Checklist

1. Effective abstraction recovery
   - Graceless failures for unstructured programs
     • break, continue, and goto statements
     • Failures cascade to large subgraphs
Unrecovered Structure

Fix: New structuring algorithm featuring Iterative Refinement

This break edge prevents progress

s1;
while (e1) {
    if (e2) { break; }
}s2;
s3;
L1: if (e1) { goto L2; }
else { goto L4; }
L2: if (e2) { goto L4; }
L3: s2; goto L1;
L4: s3;
Iterative Refinement

- Remove edges that are preventing a match
  - Represent in decompiled source as `break`, `goto`, `continue`

- Allows structuring algorithm to make more progress
Iterative Refinement

Original

```c
s1;
while (e1) {
    if (e2) { break; }
    s2;
}
s3;
```

Decompiled

```c
s1;
while (e1) {
    if (e2) { break; }
    s2;
}
s3;
```
Structural Analysis Property Checklist

1. Effective abstraction recovery
   - Graceless failures for unstructured programs
     • break, continue, and gotos
     • Failures cascade to large subgraphs

2. Correctness
1. Effective abstraction recovery
   - Graceless failures for unstructured programs
     - break, continue, and goes
     - Failures cascade to large subgraphs

2. Correctness
   - Not originally intended for decompilation
   - Structure can be incorrect for decompilation
Fix: Ensure patterns are Semantics Preserving

```java
while (true) {
    s1; if (x==1) goto L2;
    if (y==2) goto L1;
}
```
Semantics Preservation

- Applies inside of control flow structuring too
Phoenix Implementation and Evaluation
```c
int f (void) {
    int a = 42;
    int b = 0;
    while (a) {
        if (b) {
            puts("c");
            break;
        } else {
            puts("d");
        }
        a--;
        b++;
    }
    puts ("e");
    return 0;
}
```

```c
int f (void) {
    int a = 42;
    int b = 0;
    while (a) {
        if (b) {
            puts("c");
            break;
        } else {
            puts("d");
        }
        a--;
        b++;
    }
    puts ("e");
    return 0;
}
```
Large Scale Experiment Details

• Decompilers tested
  – Phoenix
  – Hex-Rays (industry state of the art)
  – Boomerang (academic state of the art)

• Boomerang
  • Did not terminate in <1 hour for most programs

• GNU coreutils 8.17, compiled with gcc
  – Programs of varying complexity
  – Test suite
Metrics (end-to-end decompiler)

1. Effective abstraction recovery
   – Control flow structuring

2. Correctness
Control Flow Structure:
Gotos Emitted (Fewer Better)

Phoenix: 40
Hex-Rays: 51
Control Flow Structure:
Gotos Emitted (Fewer is Better)

1229

Phoenix: 40
Phoenix (orig. structural analysis): 1229
Hex-Rays: 51
Are these two programs semantically equivalent?
Scalable: Testing

Original Source

int f (int x) {
    int y = 1;
    while (x > y) {
        y++;
    }
    return y;
}

Passes tests

Recovered Source

int f (int a) {
    int v = 1;
    while (a > v++)
    {
    }
    return v;
}

Passes tests

Is the decompiled program consistent with test requirements?
Number of Correct Utilities

All Utilities: 107

- Hex-Rays: 28
- Phoenix: 60
Correctness

• All known correctness errors attributed to type recovery
  – No known problems in control flow structuring

• Rare issues in TIE revealed by Phoenix stress testing
  – Even one type error can cause incorrectness
  – Undiscovered variables
  – Overly general type information
Conclusion

• Phoenix decompiler
  – Ultimate goal: Correct, abstract decompilation
  – Control-flow structuring algorithm
    • Iterative refinement
    • Semantics preserving schemas

• End-to-end correctness and abstraction recovery experiments on >100 programs
  – Phoenix
    • Control flow structuring: 😊
    • Correctness: 50% 😞

• Correct, abstract decompilation of real programs is within reach
  – This paper: improving control flow structuring
  – Next direction: improved static type recovery
Thanks! 😊

• Questions?

Edward J. Schwartz
edmcmman@cmu.edu
http://www.ece.cmu.edu/~ejschwar
END