# Static Analysis of Binaries for Reverse Engineering

Alexander Sepp, Bogdan Mihaila, Axel Simon

## Disassembling and Control Flow Reconstruction

- Code vs. data separation problem while disassembling binaries
- Resolve indirect jumps (e.g. from switch construct, function pointers)
- ⇝ interleave value analysis with disassembling

## RREIL (Relational Reverse Engineering Intermediate Language)

Example loop in C:

```
for (int i = -1; i > -100; i--)
{
/* body */
}
```

x86-64 translation:

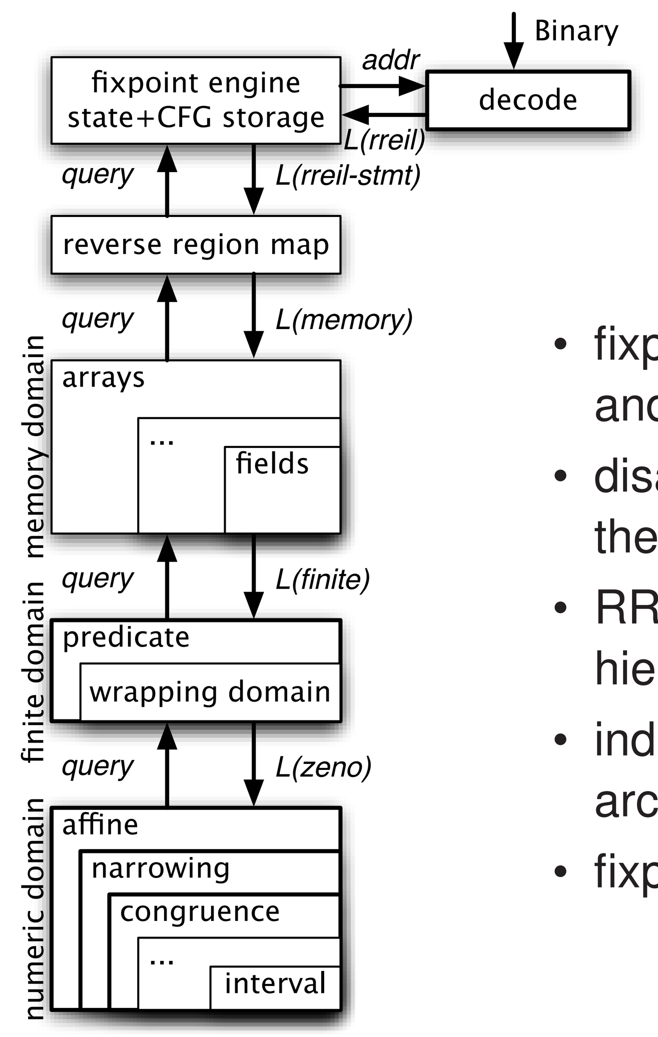
```
01: mov eax,0xffffffff
02: sub eax,0x1
/* body */
03: cmp eax,0xffffff9c
04: jg 02
```

RREIL translation:

```
01.00: mov rax:32, -1:32
01.01: mov rax:32/32, 0:32
02.00: sub t0:32, rax:32, 1:32
02.01: cmpltu CF:1, rax:32, 1:32
02.02: cmpleu BE:1, rax:32, 1:32
02.03: cmplts LT:1, rax:32, 1:32
02.04: cmples LE:1, rax:32, 1:32
02.05: cmpeq ZF:1, rax:32, 1:32
02.06: cmplts SF:1, t0:32, 0:32
02.07: xor OF:1, LT:1, SF:1
02.08: mov rax:32, t0:32
02.09: mov rax:32/32, 0:32
/* body */
03.00: sub t0:32, rax:32, -100:32
03.01: cmpltu CF:1, rax:32, -100:32
03.02: cmpleu BE:1, rax:32, -100:32
03.03: cmplts LT:1, rax:32, -100:32
03.04: cmples LE:1, rax:32, -100:32
03.05: cmpeq ZF:1, rax:32, -100:32
03.06: cmplts SF:1, t0:32, 0:32
03.07: xor OF:1, LT:1, SF:1
04.00: xor t0:1, LE:1, 1:1
04.01: brc t0:1, 02.00
```

- use small architecture independent, intermediate language for analysis
- introduce *virtual* flags `BE`,`LT`,`LE` to associate comparison operation with test used at the jump (*liveness analysis* discards unneeded flags)
- use test and *relational information* to infer bounds for variable values

## The Analyzer Structure




- fixpoint engine is the driver for the disassembler and the domain hierarchy
- disassembles a chunk of bytes and translates them to RREIL instructions
- RREIL instructions are processed by the domain hierarchy (down channel)
- indirect jumps are resolved by querying the hierarchy (up channel)
- fixpoint continues disassembly at jump target

## Memory Domain

From operations on registers/memory $m \in \mathbb{M}$ to operations on numeric variables $x \in \mathbb{X}$:

Consider x86-64: `mov eax,0xffffffff`

- 32-bit `mov` implicitly clears upper 32 bits of `rax` (the `eax` register comprises the lower 32-bits of the `rax` register)
- cannot store -1 in numeric domain since then `rax=0xffffffffffffffff`
- tracking `rax` as one numeric variable leads to precision loss
- *idea:* use fields with size and offset in RREIL

```
mov rax:32/0, -1:32
mov rax:32/32, 0:32
```

| rax:32/32 | rax:32/0 |
|---|---|
| 0 | -1 |

## Dealing with Finite Integer Arithmetic

We have a special view on numeric information:

- we store e.g. `al` $\in [254, 257]$ (`al` is 8 bits big)
- this means:

$$\begin{aligned} 254 &\equiv 11111110 \\ 255 &\equiv 11111111 \\ 256 &\equiv 00000000 \\ 257 &\equiv 00000001 \end{aligned}$$

- which is also $[254, 255] \sqcup [0, 1] = [0, 255]$
- we call this conversion *wrapping*

The finite domains associate a bit-size with each $x \in \mathbb{X}$.

- *idea:* wrap operand `i` before each operation; wrapping is no-op if `i`$\in [-2^{31}, 2^{31} - 1]$
- *problem:* precision loss; `add`, `sub` carry no sign information (signedness-agnostic)
- *idea:* only wrap when unavoidable, e.g. before executing the test `i > -100`
- *problem:* `i` $\in [-1, -1], [-2, -1], [-3, -1], [-4, -1] \ldots$ is inferred during fixpoint computation; *widening* applied to `i` $\rightsquigarrow [-\infty, -1]$
- ⇝ wrapping of widened value $[-\infty, -1]$ gives $[-2^{31}, 2^{31} - 1]$
- cannot infer that `i` is negative

## Narrowing Domain

Avoiding precision loss incurred by widening:

- have a narrowing domain that tracks all tests that don't affect the state (redundant tests)
- ⇝ the test `i > -100` is stored when analyzing the loop
- after widening `i` to $[-\infty, -1]$ apply all stored tests
- ⇝ `i` $\in [-99, -1]$ follows
- wrapping to positive values avoided

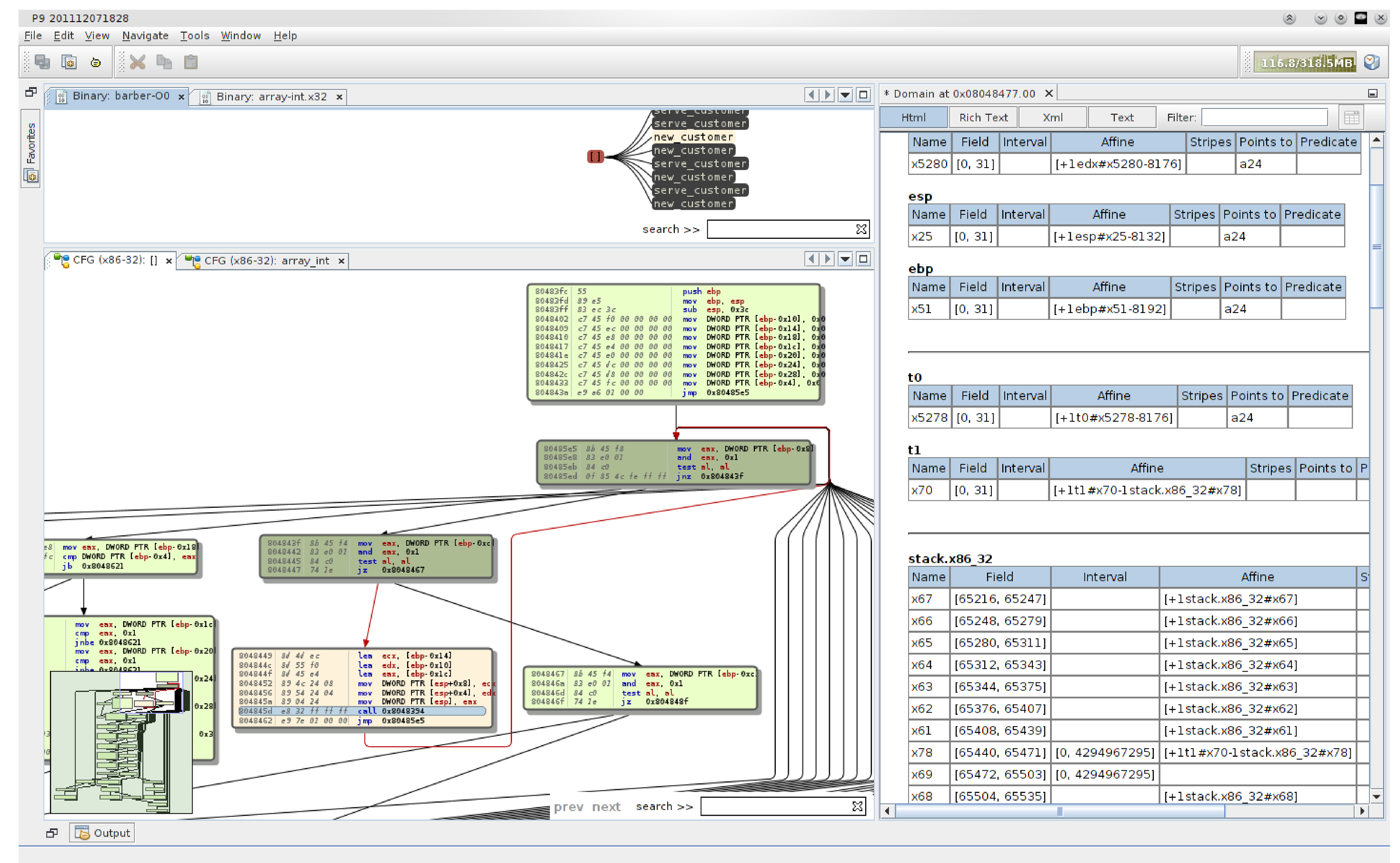
## Numeric Domain

Numeric domains map variables $x \in \mathbb{X}$ to a subset of $\mathbb{Z}$.

- the affine tracks equalities $c_i x_i = \sum_j c_j x_j$ where $i < j$
- ⇝ no need to store $x_i$ in child domains; some linear assignments need not be propagated to child
- maintains equalities between memory and register locations when inferring numerical values
- the interval domain maps $x_k$ to $[l_k, u_k]$

## Tool Implementation

The mentioned techniques have been implemented as an analyzer tool:



- able to disassemble Linux (ELF) and Windows (PE) binaries
- displays RREIL and native Control Flow Graphs for each discovered procedure
- shows inferred domain values for the variables at each program point
- shows warnings that occurred during the program analysis

## Publications


[1] B. Mihaila, A. Sepp, and A. Simon. Widening as Abstract Domain. In *NASA Formal Methods*, volume 7871 of *LNCS*, pages 170–186, Moffett Field, California, USA, May 2013. Springer.

[2] A. Sepp, B. Mihaila, and A. Simon. Precise Static Analysis of Binaries by Extracting Relational Information. In M.Pinzger and D. Poshyvanyk, editors, *Working Conference on Reverse Engineering*, Limerick, Ireland, October 2011. IEEE Computer Society.

[3] H. Siegel, B. Mihaila, and A. Simon. The Undefined Domain: Precise Relational Information for Entities that Do Not Exist. In C. Shan, editor, *Asian Symposium on Programming Languages and Systems*, Melbourne, Australia, December 2013. Springer.