A Semantics-Based Approach to Concept Assignment in Assembly Code

Zachary Sisco

ICCWS 2017, Dayton, Ohio, USA

March 3, 2017

The Concept Assignment Problem

"Recognizing concepts and assigning them to locations within a program in order to build an understand of that program"

- Programming constructs, program features, behaviors
- Extensively researched in source code but not *binary code*

The Concept Assignment Problem in *Assembly*

Why?

- Reverse engineers read assembly code to understand binaries
- Cyber analysts use tools to understand behaviors of vulnerable or malicious software
- Enable automated approaches to describe and discover classes of vulnerabilities in software

Specify Formal Language and Semantics

Lift assembly to an Intermediate Language

$program$	$::=$	$stmt$*
$stmt\ s$	$::=$	$var\ := exp$ | store(exp, exp) | goto exp | assert exp | if exp then goto exp else goto exp | halt
$exp\ e$	$::=$	load(exp) | $exp \diamond_b exp$ | $\diamond_u exp$ | var | get_input(src) | v
$\diamond_b$	$::=$	$+ \mid - \mid * \mid / \mid \vee \mid \wedge \mid < \mid \leq \mid > \mid \geq \mid =$
$\diamond_u$	$::=$	$-$ (unary minus) | $\neg$ (logical negation)
$value\ v$	$::=$	32-bit unsigned integer | $\perp$
src	$::=$	string

Figure 1: Grammar of a simple intermediate language

Example - Dynamic Memory Allocation

C Source Code

```c
int main() {
    int *heap_array = (int*)malloc(3 * sizeof(int));
    int index = 2;
    heap_array[0] = 10;
    heap_array[1] = 20;
    heap_array[index] = 30;
}
```

Assembly

```
0000000000400506 <main>:
  400506:        push   rbp
  400507:        mov    rbp,rsp
  40050a:        sub    rsp,0x10
  40050e:        mov    edi,0xc
  400513:        call   400400 <malloc@plt>
  400518:        mov    QWORD PTR [rbp-0x8],rax
  40051c:        mov    DWORD PTR [rbp-0xc],0x2
  400523:        mov    eax,QWORD PTR [rbp-0x8]
  400527:        mov    DWORD PTR [rax],0xa
  40052d:        mov    rax,QWORD PTR [rbp-0x8]
  400531:        add    rax,0x4
  400535:        mov    DWORD PTR [rax],0x14
  40053b:        mov    eax,DWORD PTR [rbp-0xc]
  40053e:        cdqe
  400540:        lea    rdx,[rax*4+0x0]
  400548:        mov    rax,QWORD PTR [rbp-0x8]
  40054c:        add    rax,rdx
  40054f:        mov    DWORD PTR [rax],0x1e
  400555:        leave
  400556:        ret
```

Intermediate Language

```
esp := esp - 1
store(esp, ebp)
ebp := esp
esp := esp - 2
edi := 3
eax := get_input(malloc)
store(ebp - 1, eax)
store(ebp - 2, 2)
eax := load(ebp - 1)
store(eax, 10)
eax := load(ebp - 1)
eax := eax + 1
store(eax, 20)
eax := load(ebp - 2)
edx := eax * 1 + 0
eax := load(ebp - 1)
eax := eax + edx
store(eax, 30)
esp := ebp
ebp := load(esp)
halt
```

Specify Formal Language and Semantics

Use operational semantics to prove the existence of a concept in a program

$$\frac{v \text{ is input from } src \qquad H'_\mu = H_\mu[v \leftarrow P_{\mathsf{malloc}}(src)] \quad H'_\rho = H_\rho[v \leftarrow \rho_{\mathsf{malloc}}(src, edi)]}{H_\mu, H_\rho, \mu, \Delta \vdash \mathsf{get\_input}(src) \Downarrow v}\ \textsc{H-Input}$$

$$\frac{\mu, \Delta \vdash e \Downarrow v \quad v' = \mu[v] \quad \varphi = \Phi(v, H_\mu[v'])}{H_\mu, \mu, \Delta \vdash \mathsf{load}\ e \Downarrow v'}\ \textsc{H-Load}$$

$$\frac{P_{\mathsf{heapcheck}}(v_1, \mu[\varphi], H_\rho[\mu[\varphi]], H_\mu[\mu[\varphi]]) = \mathbf{T} \qquad \mu, \Delta \vdash e_1 \Downarrow v_1 \quad \mu, \Delta \vdash e_2 \Downarrow v_2 \quad \mu' = \mu[v_1 \leftarrow v_2] \quad \iota = \Sigma[pc + 1]}{H_\mu, H_\rho, \Sigma, \mu, \Delta, pc, \mathsf{store}(e_1, e_2) \rightsquigarrow H_\mu, H_\rho, \Sigma, \mu', \Delta, pc + 1, \iota}\ \textsc{H-Store}$$

Figure 2: Operational semantics that trace dynamic allocation of memory

Formal Semantics

Extract information about the feature.

- How much memory was allocated to the program heap;
- The pointer in memory to the data structure;
- The total number of elements possible to allocate;
- The data stored in the array;
- If any access to the heap array is out-of-bounds (possible buffer overflow).

Integrate with Data Model

Knowledge Representation Data Model

- Organize concepts
- "Back-end" for the formal language
- Compose basic analyses together to map more complex behaviors to locations in program

Integrate with Data Model

Category-Theoretic Data Model

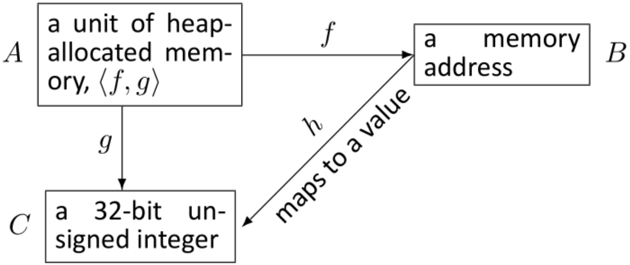

Figure 3: Data model representing a block of heap memory with address and value

Conclusions

- Re-frame and formalize concept assignment for binaries
- Method is limited by the richness of the knowledge base
- Goal: automate comprehension of binary code to support cyber analysts and reverse engineers

Thank You

Questions?