Binary inference of Typed Assembly Language

Carlo Angiuli (cangiuli@cs.cmu.edu)
Matthew Maurer (mmaurer@ece.cmu.edu)

March 22, 2012

Project website: http://www.cs.cmu.edu/~cangiuli/15745

Description

Type reconstruction on binary code is useful for recovering high-level structure (e.g., for reverse engineering) and for verifying basic safety properties (e.g., we do not dereference non-pointer values), especially when a program’s source code is not present. There are a number of techniques for binary type reconstruction, including observation of dynamic memory access patterns [6], and value-set analysis [1], a static dataflow analysis.

Prof. David Brumley’s research group is developing BAP [2], a binary analysis platform which includes limited type inference [4]. The type system that has been developed for BAP’s intermediate representation (which elaborates x86 instructions to a related, simpler language) has a hierarchy of sized integer, pointer, record, and function types.

However, these projects all reconstruct relatively simple types. In contrast, the Typed Assembly Language project [5] is a family of very expressive type systems for realistic assembly languages, including such features as stack-based reasoning, polymorphism, and existential types. These type systems permit high-level reasoning, such as reachability of heap objects for garbage collection, and validity of stack accesses [3].

Our proposal is to extend BAP’s type inference to a richer type system styled after TAL. We plan to include such features as typestate for heap pointers, to detect, e.g., use-after-free bugs; array bounds checks; and allowing user-supplied type annotations to facilitate more precise type inference.

After these basic features are in place, we hope to extend our analysis to be robust in the presence of more complex control flow, and to optionally check adherence to the ABI, e.g., that caller stack frames are not disturbed.

Our most ambitious goal would be to suggest type annotations to the user, i.e., to determine assertions which cannot be inferred from the code, but which would allow checking much stronger safety properties of the code. These additional annotations could then be transformed into runtime assertions.
Logistics

We already have access to the BAP source repository, and have communicated with Prof. Brumley, who believes our project would be a valuable extension to BAP. Our next goal is to better familiarize ourselves with the TAL literature, and with the BAP type inferencing code.

Plan of Attack

<table>
<thead>
<tr>
<th>Week</th>
<th>Maurer</th>
<th>Carlo</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reading TAL + VSA, reviewing TIE code</td>
<td>Reading TAL, reading TIE code, BAP familiarization</td>
</tr>
<tr>
<td>2</td>
<td>Adapt TAL, straightline rules (heap)</td>
<td>Adapt TAL, straightline rules (heap)</td>
</tr>
<tr>
<td>3</td>
<td>Improve BAP VSA, use it to reify CFG</td>
<td>Extend to direct, non-cyclic control flow</td>
</tr>
<tr>
<td>4</td>
<td>Cyclic control flow</td>
<td>Cyclic control flow</td>
</tr>
<tr>
<td>5</td>
<td>Teach it stack discipline and ABI</td>
<td>Teach it stack discipline and ABI</td>
</tr>
<tr>
<td>6</td>
<td>Assertion recommendation</td>
<td>Assertion recommendation</td>
</tr>
</tbody>
</table>

Be aware that some weeks (in particular week 2) may actually expand a little bit compared to others, so this may not be an exact schedule.

Milestone

By the milestone meeting, we intend to have an inference engine able to type annotate heap-based memory usage over non-cyclic, direct control flow, and some cases of cycles. This will represent the meat of the work. We also expect to be able to speak about stacks at this point, however there will be no notion of "deletion" or "allocation", a la malloc and free yet.

Literature Search

We have already collected most of the necessary papers. A deeper reading of VSA will likely be needed to improve BAP’s existing VSA support, and our type system will likely be very similar to TAL, so more thorough reading is needed there as well. Balakrishnan’s thesis appears to be the state of the art in VSA (even though two years old), and TAL appears to be the state of the art for types on x86 assembly, so we feel that we have a good basis.

Resources Needed

We already have a copy of the necessary software (BAP), and commit access. Other software is essentially incidental. We do not need any specialized hardware, as our analysis is primarily a correctness analysis. As a result, our personal machines are acceptable hardware.

Getting Started

Thus far, we have created the appropriate repositories and branches to begin work, and begun reading papers. Nothing really prevents us from starting immediately.

References


