Huan Nguyen
Secure System Lab
Date: 12/12/2018

Targets

What is finished:
- A gcc plugin to extract RTL when compiling packages
- A code to generate *.imap files based on output of gcc plugin
- Modify lex/parse of lifting code to support x64 system
- Modify disassembly.sh to support x64 system

What is on going:
- A translator from RTL source to prolog for type inference problem

What to be fixed:
- Fix the lifting code to support sequence of at least 5 asm instructions (previously up to 4)
- Fix the lifting code that could limit capacity of pair <asm, rtl>

Extracting RTL during compilation

The gcc compiler in Makefile of any package must be replaced by this plugin in order to produce pairs of assembly and RTL code during compilation.

A technical problem in crafting this plugin is in printing asm instructions. They are exported to a temporary file which is removed instantly after. My first approach is redirecting asm_out_file to another file for each asm instruction so that I can collect it easily, but then it raises the error "too many open files". The next approach is to wait until all asm instructions are printed to asm_out_file and then open it and write one by one to another file, but parts of them are incorrect. Our solution is to create a hard link to the asm_out_file and also printing the RTL to asm_out_file. Doing so will systematically prevents any attempt to intercept the temporary file.

The output this plugin produces is a directory of files (e.g. tmp_1.txt, tmp_2.txt, ...) and each of them contains multiple pairs of <asm, rtl> of a function. This plugin supports both x86 and x64 system.

Generating *.imap files

The purpose of this part is to produce the input for the learning phase of LISC [1]. It's written in C++ and organized as following:

- *format_asm*: provide tools to refine asm instruction
- *format_rtl*: provide tools to refine rtl instruction, eliminate redundant information
- *x64*: pass label from asm to rtl, synchronize symbol_ref in asm and rtl, rectify register names, use *format_rtl* and *format_asm* to refine and standardize instructions
- *main*: go through all tmp_i.txt files in a specific directory

This code supports both x86 and x64 system. In addition, we can use 2 UNIX commands to eliminate duplicates in *.imap file before passing to LISC [1].

Support for x64 systems

Previously, the old disassembly code which was built on top of objdump works only for x86. By looking at the disassembly output for "ls", I can identify two purposes of the shellscript:
- Match the opcode of asm produced by objdump and gcc
- Standardize the format for LISC to lift up to RTL
I made some necessary changes into the shellscript to make it work with x64 system.

Moreover, based on parseX86.mly, I wrote a new file parseX64.mly in LISC to:
- Support one more case in definition of "arg" to handle symbol_ref in x64
- Support new approach to related opcodes: "movl", "movq" share the same root "mov", the suffix indicates the mode it operates in; thus to connect them, we split the suffix and view it as an operand. For example, "movq $0, 0(%r13)" is parsed into "mov_3 q, $0, 0(%r13)", where 3 is number of arguments. Moreover, to those opcodes that don't have suffix such as "call", we use "_" as the suffix operand, e.g. "call *104(%r10)" is parsed into "call_2 _,*104(%r10)".

Type Inference Problem

In the scope of our research, we'd like to identify which and where a register possibly hold an address instead of a value. If we know at a specific point a register might holds an address, we can add only those register values we analyzed into the set of valid targets. This is often useful for a more fine-grained policy for Control Flow Integrity on binary code.
Usually, value which a register holds is a number. The main idea is: by traversing the flow, if a value is possibly dereferenced somewhere, we can say it's a potential address. In order to efficiently analyze type inference, we make use of XSB Prolog, which supports fixed point iteration automatically to avoid eternal loops. The code is written in C++ and organized as following:

- *rtl*: provide tools to process a single RTL instruction, using *mathexpr* for estimate_val method
- *mathexpr*: convert an expression in RTL to prefix-order expression, using class Expr in *rtl*
- *basicblock*: provide tools to process a basic block, using *rtl*
- *translator*: output prolog code to analyze type of value each register holds, using *basicblock*

This code is incomplete. Specifically, codes in *mathexpr* and *basicblock* are completed while codes in *mathexpr* and *translator* are being added more features. I postponed it for reading papers to gain more background in the field.

What to be fixed

In x64 system, one RTL instruction could represent for at least 5 asm instructions. These instructions are found in exactly same order after dissasembly using objdump. Because the current code only supports up to 4 asm instructions, *main.ml* should be fixed.

The learning code works for individual *.imap file: x64.openssl.imap, x64.binutils.imap, x64.glibc.imap.

However, an issue occurs when it takes a combination of these *.imap files and leads to 0% success rate. Therefore, I think the code might have problem with capacity. Moreover, capacity is not necessary number of pairs in input. It's possibly the number of entry stored following the learning algorithm.

Another obstruction that also prevents replicating previous experiments is that for some packages, I cannot find where to replace the plugin into. Another problem is that some tmp_i.txt are missing although file tmp_Rtl.txt, which stores temporary RTL instructions, continuously changes in size.

References

[1] Niranjan Hasabnis and R. Sekar. *Lifting Assembly to Intermediate Representation: A Novel Approach Leveraging Compilers*. ASPLOS 2016.