

Proof Generating from IL to x86 (SCAP)

Technique Report

Zhaopeng Li Zhenting Zhang Yiyun Chen

School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230026, China
 Software Security Lab., Suzhou Institute for Advanced Study, University of Science and Technology of China,
 Suzhou, Jiangsu 215123, China
 {zpli}@mail.ustc.edu.cn yiyun@ustc.edu.cn

Abstract

Proof generation is one of the important phases in certifying compilation. Codes (in our intermediate language) and assertions have been outputted by the front-end of our certifying compiler CComp. In this article, proof generation approach will be presented from intermediate language to x86 assembly language using SCAP as program verification framework. It includes core algorithms, data structures and examples.

Keywords Software Safety, Hoare Logic, Separation Logic, Proof Generation, Certifying Compiler

1. Introduction

Proof generation is one of the important phases in certifying compilation. Codes (in our intermediate language) and assertions have been outputted by the front-end of our certifying compiler CComp. In this report, approaches will be introduced for proof generating from our intermediate language to x86 assembly languages using SCAP as program verification framework.

This report presents the approach of proof generation. The rest is organized as following: section 2 presents our version of SCAP for x86 assembly language; section 3 presents code and assertion translating and generating from IL to x86; section 4 gives some examples; and section 5 concludes.

2. Assembly Program Verification Framework SCAP

2.1 Abstract Machine m86

We focus on code compiled by CComp, so self-modified code is not considered. That is, the memory resident by code will not change during program execution. So we separate this portion of memory from the data portion (\mathbb{M}) using code heap (\mathbb{C}). In figure1, the x86-style abstract machine is defined.

A world \mathbb{W} of m86 includes code heap \mathbb{C} , machine state \mathbb{S} and a basic block \mathbb{I} in execution. Note that, there is no program counter (pc) in the abstract machine, we use this basic block to simulate it.

(World)	$\mathbb{W} ::= (\mathbb{C}, \mathbb{S}, \mathbb{I})$
(Code Heap)	$\mathbb{C} ::= \{l \sim \mathbb{I}\}^*$
(State)	$\mathbb{S} ::= (\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K})$
(Data Heap)	$\mathbb{H} ::= \{l \sim w\}^*$
(Stack)	$\mathbb{K} ::= (\mathbb{W}_{bp}, \mathbb{W}_{sp}, \mathbb{W} :: \dots :: \text{knil})$
(Register File)	$\mathbb{R} ::= \{r \sim w\}^*$
(Flag Reg. File)	$\mathbb{R}_f ::= \{\text{flag} \sim b\}^*$
(Register)	$r ::= \text{eax} \mid \text{ebx} \mid \text{ecx} \mid \text{edx} \mid \text{edi} \mid \text{esi}$
(Flag Register)	$\text{flag} ::= \text{zf} \mid \text{sf}$
(Address)	$a ::= (i) \mid i(r)$
(Instruction)	$c ::= \begin{array}{l} \text{addir } i, r_d \mid \text{addr } r_s, r_d \\ \text{subir } i, r_d \mid \text{subr } r_s, r_d \\ \text{mulir } i, r_d \mid \text{mulr } r_s, r_d \\ \text{movir } i, r_d \mid \text{movr } r_s, r_d \\ \text{movrm } r_s, a \mid \text{movm } a, r_d \\ \text{movkr } i, r_d \mid \text{movrk } r_s, i \\ \text{pushi } i \mid \text{pushr } r_s \mid \text{popr } r_d \\ \text{cmpi } i, r_d \mid \text{cmpr } r_s, r_d \\ \text{je } l \mid \text{jne } l \mid \text{jg } l \mid \text{jge } l \\ \text{enter } i \mid \text{leave} \\ \text{free } r_s \mid \text{malloc } r_s \end{array}$
(Basic Block)	$\mathbb{I} ::= c; \mathbb{I} \mid \text{jmp } l \mid \text{ret } i \mid \text{call } f, l$
(Labels, Word)	$l, f, w ::= i(\text{integer})$
(Bit)	$b ::= 0 \mid 1$

Figure 1. Abstract Machine m86

The operational semantics of m86 (in Figure2) is modeled as small-step transition between worlds.

2.2 Specification

(Assertion)	$p \in \text{State} \rightarrow \text{Prop}$
(Guarantee)	$g \in \text{State} \rightarrow \text{State} \rightarrow \text{Prop}$
(Code Spec.)	$a ::= (p, g)$
(Code Heap Spec.)	$\Psi ::= \{l \sim a\}^*$

Figure 3. Specifications

The specification is defined in Figure3. We use the meta-logic (Prop) of Coq as our assertion language.

Each basic block is specified by a pair of assertion and guarantee. Assertion p is predicate taking current state as parameter which means current state must satisfy predicate p . Guarantee g takes two states as parameter: current state and the state at the return point

if $\mathbb{I} =$	$(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}), \mathbb{I}) \mapsto$
jmp l	$(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}), \mathbb{C}(l))$ when $l \in \text{dom}(\mathbb{C})$
ret i	$(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}'), \mathbb{C}(l))$ when $\mathbb{K} = (w_{bp}, w_{sp}, l :: \text{data})$ and $l \in \text{dom}(\mathbb{C})$ and $\mathbb{K}' = (w_{bp}, w_{sp} + 4 + 4 * i, \text{popSK}(\text{data}, 4 + 4 * i))$
call f, l	$(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}'), \mathbb{C}(f))$ when $\mathbb{K} = (w_{bp}, w_{sp}, \text{data})$ and $f, l \in \text{dom}(\mathbb{C})$ $w_{sp} - 4 > 0$ and $\mathbb{K}' = (w_{bp}, w_{sp} - 4, l :: \text{data})$
je $l; \mathbb{I}'$	$(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{I}')$ when $\mathbb{R}_f(\mathbf{zf}) = 1$ $(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{C}(l))$ when $\mathbb{R}_f(\mathbf{zf}) = 0$ and $l \in \text{dom}(\mathbb{C})$
jne $l; \mathbb{I}'$	$(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{I}')$ when $\mathbb{R}_f(\mathbf{zf}) = 0$ $(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{C}(l))$ when $\mathbb{R}_f(\mathbf{zf}) = 1$ and $l \in \text{dom}(\mathbb{C})$
jl $l; \mathbb{I}'$	$(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{I}')$ when $\mathbb{R}_f(\mathbf{zf}) = 1$ or $\mathbb{R}_f(\mathbf{sf}) = 1$ $(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{C}(l))$ when $\mathbb{R}_f(\mathbf{zf}) = 0$ and $\mathbb{R}_f(\mathbf{sf}) = 0$ and $l \in \text{dom}(\mathbb{C})$
jge $l; \mathbb{I}'$	$(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{I}')$ when $\mathbb{R}_f(\mathbf{sf}) = 1$ $(\mathbb{C}, (\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{C}(l))$ when $\mathbb{R}_f(\mathbf{sf}) = 0$ and $l \in \text{dom}(\mathbb{C})$
c; \mathbb{I}'	$(\mathbb{C}, \text{Next}_c(\mathbb{H}, \mathbb{R}, \mathbb{R}_f), \mathbb{I}')$

where

if $c =$	$\text{Next}_c(\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}) =$
addi i, r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow \mathbb{R}(r_d) + i\}, \mathbb{R}_f, \mathbb{K})$
addr r_s, r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow \mathbb{R}(r_d) + \mathbb{R}(r_s)\}, \mathbb{R}_f, \mathbb{K})$
subir i, r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow \mathbb{R}(r_d) - i\}, \mathbb{R}_f, \mathbb{K})$
subrr r_s, r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow \mathbb{R}(r_d) - \mathbb{R}(r_s)\}, \mathbb{R}_f, \mathbb{K})$
mulir i, r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow \mathbb{R}(r_d) * i\}, \mathbb{R}_f, \mathbb{K})$
movir i, r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow i\}, \mathbb{R}_f, \mathbb{K})$
movrr r_s, r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow \mathbb{R}(r_s)\}, \mathbb{R}_f, \mathbb{K})$
movrm $r_s, (i)$	$(\mathbb{H}\{i \rightsquigarrow \mathbb{R}(r_s)\}, \mathbb{R}, \mathbb{R}_f, \mathbb{K})$
movrm $r_s, i(r_d)$	$(\mathbb{H}\{\mathbb{R}(r_d) + i \rightsquigarrow \mathbb{R}(r_s)\}, \mathbb{R}, \mathbb{R}_f, \mathbb{K})$ when $\mathbb{R}(r_d) + i \in \text{dom}(\mathbb{H})$
movmr $(i), r_d$	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow \mathbb{H}(i)\}, \mathbb{R}_f, \mathbb{K})$ when $i \in \text{dom}(\mathbb{H})$
movmr $i(r_s), r_d$	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow \mathbb{H}(\mathbb{R}(r_s) + i)\}, \mathbb{R}_f, \mathbb{K})$ when $\mathbb{R}(r_s) + i \in \text{dom}(\mathbb{H})$
movkr i, r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow w\}, \mathbb{R}_f, \mathbb{K})$ when $w = \text{readSK}(\mathbb{K}, i)$
movrk r_s, i	$(\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}')$ when $\mathbb{K}' = \text{updSK}(\mathbb{K}, i, \mathbb{R}(r_s))$
pushi i	$(\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}')$ when $\mathbb{K} = (w_{bp}, w_{sp}, \text{data})$ $w_{sp} - 4 > 0$ and $\mathbb{K}' = (w_{bp}, w_{sp} - 4, i :: \text{data})$
pushr r_s	$(\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}')$ when $\mathbb{K} = (w_{bp}, w_{sp}, \text{data})$ $w_{sp} - 4 > 0$ and $\mathbb{K}' = (w_{bp}, w_{sp} - 4, \mathbb{R}(r_s) :: \text{data})$
popr r_d	$(\mathbb{H}, \mathbb{R}\{r_d \rightsquigarrow w, \mathbb{R}_f, \mathbb{K}'\})$ when $\mathbb{K} = (w_{bp}, w_{sp}, w :: \text{data})$ and $\mathbb{K}' = (w_{bp}, w_{sp} + 4, \text{data})$
enter i	$(\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}')$ when $\mathbb{K} = (w_{bp}, w_{sp}, \text{data})$ $w_{sp} - 4 * i - 4 > 0$ and $\mathbb{K}' = (w_{sp} - 4, w_{sp} - 4 * i - 4, \underbrace{\dots}_{i} :: w_{bp} :: \text{data})$
leave	$(\mathbb{H}, \mathbb{R}, \mathbb{R}_f, \mathbb{K}')$ when $\mathbb{K} = (w_{bp}, w_{sp}, \text{data})$ and $bp_{old} = \text{readSK}(\mathbb{K}, 0)$ $\mathbb{K}' = (bp_{old}, w_{bp} + 4, \text{popSK}(\text{data}, w_{bp} - w_{sp}))$
cmpi i, r_d	$(\mathbb{H}, \mathbb{R}, \mathbb{R}_f\{\mathbf{zf} \rightsquigarrow 1\}\{\mathbf{sf} \rightsquigarrow 0\}, \mathbb{K})$ when $\mathbb{R}(r_d) = i$ $(\mathbb{H}, \mathbb{R}, \mathbb{R}_f\{\mathbf{zf} \rightsquigarrow 0\}\{\mathbf{sf} \rightsquigarrow 0\}, \mathbb{K})$ when $\mathbb{R}(r_d) > i$ $(\mathbb{H}, \mathbb{R}, \mathbb{R}_f\{\mathbf{zf} \rightsquigarrow 0\}\{\mathbf{sf} \rightsquigarrow 1\}, \mathbb{K})$ when $\mathbb{R}(r_d) < i$
cmpr r_s, r_d	$(\mathbb{H}, \mathbb{R}, \mathbb{R}_f\{\mathbf{zf} \rightsquigarrow 1\}\{\mathbf{sf} \rightsquigarrow 0\}, \mathbb{K})$ when $\mathbb{R}(r_d) = \mathbb{R}(r_s)$ $(\mathbb{H}, \mathbb{R}, \mathbb{R}_f\{\mathbf{zf} \rightsquigarrow 0\}\{\mathbf{sf} \rightsquigarrow 0\}, \mathbb{K})$ when $\mathbb{R}(r_d) > \mathbb{R}(r_s)$ $(\mathbb{H}, \mathbb{R}, \mathbb{R}_f\{\mathbf{zf} \rightsquigarrow 0\}\{\mathbf{sf} \rightsquigarrow 1\}, \mathbb{K})$ when $\mathbb{R}(r_d) < \mathbb{R}(r_s)$

Figure 2. Operation Semantics of m86

of the current function (if it ever returns). So guarantee is used to describe the relation between these two states.

2.3 Inference Rules

We use the following judgements to define inference rules:

- $\Psi \vdash \{a\}\mathbb{W}$ (well-formed world)
- $\Psi \vdash C : \Psi'$ (well-formed code heap)
- $\Psi \vdash \{a\}\mathbb{I}$ (well-formed basic block)

In Figure 4, inference rules are shown.

$\Psi \vdash \{a\}\mathbb{W}$

 (well-formed world)

$$\frac{\Psi \vdash C : \Psi \quad p \in \mathbb{S} \wedge \exists n. \text{WFST}(n, g, \mathbb{S}, \Psi) \quad \Psi \vdash \{(p, g)\}\mathbb{I}}{\Psi \vdash \{(p, g)\}(C, \mathbb{S}, \mathbb{I})} \quad (\text{WORLD})$$

$\Psi \vdash C : \Psi'$

 (well-formed code heap)

$$\frac{\Psi' \subseteq \Psi \quad \forall l \in C. l \in \text{dom}(\Psi') \rightarrow \Psi \vdash \{\Psi'(l)\}C(l)}{\Psi \vdash C : \Psi'} \quad (\text{CDHP})$$

$$\frac{\Psi_1 \vdash C_1 : \Psi'_1 \quad \Psi_2 \vdash C_2 : \Psi'_2 \quad \text{dom}(C_1) \cap \text{dom}(C_2) = \emptyset \quad \forall f \in \text{dom}(\Psi_1) \cup \text{dom}(\Psi_2). \Psi_1(f) = \Psi_2(f)}{\Psi_1 \cup \Psi_2 \vdash C_1 \cup C_2 : \Psi'_1 \cup \Psi'_2} \quad (\text{LINK})$$

$\Psi \vdash \{(p, g)\}\mathbb{I}$

 (well-formed basic block)

$$\frac{\Psi' \subseteq \Psi \quad \forall l \in C. l \in \text{dom}(\Psi') \rightarrow \Psi \vdash \{\Psi'(l)\}C(l)}{\Psi \vdash \{(p, g)\}} \quad (\text{JMP})$$

Figure 4. Inference Rules

References

- [1] G. Necula. Proof-carrying code, In *Proc. 24th ACM Symposium on Principles of Programming Languages*, pages 106-119, Jan 1997.
- [2] G. Necula, P. Lee. The Design and Implementation of a Certifying Compiler, In *Proceedings of the '98 Conference on Programming Language Design and Implementation*, Montreal, 1998.
- [3] Y. Y. Chen, L. Ge, B. J. Hua, Z. P. Li and C. Liu. Design of a Certifying Compiler Supporting Proof of Program Safety. In *Proceedings of 1st IEEE IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE 2007)*, pages 117-126, June 2007.
- [4] Y. Y. Chen, L. Ge, B. J. Hua, Z. P. Li, C. Liu and Z. F. Wang. A Pointer Logic and Certifying Compiler. *Frontiers of Computer Science in China*.1(3):297-312, July 2007.
- [5] B. J. Hua, Y. Y. Chen, L. Ge, and Z. F. Wang. The PointerC programming language specification. (*Technical Report*) Available at: <http://sbg.ustcsz.edu.cn/lss/doc/>.
- [6] J. C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55-74, July 2002.
- [7] J. C. Reynolds. An introduction to separation logic. Lecture notes available at <http://www.cs.cmu.edu/jcr/>, Spring 2005.