Symbolic Rules

Primitive Types

nop, ubranch :

$$(-, \texttt{insn}, -, -, -) \to (-, -) \quad (1)$$

ar2s1d : $(-, \texttt{insn}, \texttt{other}, \texttt{other}, -) \to (-, \texttt{other})$ (2)

ar2s1d : $(-, \texttt{insn}, \texttt{addr}, \texttt{other}, -) \to (-, \texttt{addr})$ (3)

ar2s1d : $(-, \texttt{insn}, \texttt{other}, \texttt{addr}, -) \to (-, \texttt{addr})$ (4)

ar2s1d : $(-, \texttt{insn}, \texttt{addr}, \texttt{addr}, -) \to (-, \texttt{other})$ (5)

ar1s1d : $(-, \texttt{insn}, \texttt{other}, -, -) \to (-, \texttt{other})$ (6)

ar1s1d : $(-, \texttt{insn}, \texttt{addr}, -, -) \to (-, \texttt{addr})$ (7)

ar1d, flags :

$$(-, \texttt{insn}, -, -, -) \to (-, \texttt{other}) \quad (8)$$

cbranch : $(-, \texttt{insn}, \texttt{other}, \texttt{other}, -) \to (-, -)$ (9)

ijump, return :

$$(-, \texttt{insn}, \texttt{addr}, -, -) \to (-, -) \quad (10)$$

dcall, icall :

$$(-, \texttt{insn}, \texttt{addr}, -, -) \to (-, \texttt{addr}) \quad (11)$$

load : $(-, \texttt{insn}, \texttt{addr}, -, t) \to (-, t)$ if $t \neq \texttt{insn}$ (12)

store : $(-, \texttt{insn}, t, \texttt{addr}, -) \to (-, t)$ if $t \neq \texttt{insn}$ (13)

move : $(-, \texttt{insn}, \texttt{other}, -, -) \to (-, \texttt{other})$ (14)

move : $(-, \texttt{insn}, \texttt{addr}, -, -) \to (-, \texttt{addr})$ (15)

Alternate rules for checking return-address:

ijump : $(-, \texttt{insn}, \texttt{addr}, -, -) \to (-, -)$ (10)

return : $(-, \texttt{insn}, \texttt{retaddr}, -, -) \to (-, -)$ (10)

dcall, icall :

$$(-, \texttt{insn}, \texttt{addr}, -, -) \to (-, \texttt{retaddr}) \quad (11)$$

Memory Safety

N-coloring with $N = 2^{64} - k$ for full memory safety. We write colors as c, and use them to tag pointers to the heap. We assume a special tag $\perp$ that is different than the colors, and which is used to tag all data that is not pointers to the heap. The tags for registers are colors or $\perp$ (written t). The tags for memory are pairs of a color and either a color or $\perp$ (written (c_1, t_2)) or F (unallocated). The heap is initially all tagged F. Finally the tags on instructions are drawn from the set: $\{t_{malloc}, t_{mallocinit}, t_{freeinit}, t_{something\_else}\}$.

nop, cbranch, ubranch, ijump, return :

$$(-, -, -, -, -) \to (-, -) \quad (1)$$

ar2s1d : $(-, -, \perp, \perp, -) \to (-, \perp)$ (2)

ar2s1d : $(-, -, c, \perp, -) \to (-, c)$ (3)

ar2s1d : $(-, -, \perp, c, -) \to (-, c)$ (4)

ar2s1d : $(-, -, c, c, -) \to (-, \perp)$ (5)

ar1s1d : $(-, -, t, -, -) \to (-, t)$ (6)

ar1d, dcall, icall, flags :

$$(-, -, -, -, -) \to (-, \perp) \quad (7)$$

load : $(-, -, c_1, -, (c_2, t_2)) \to (-, t_2)$ if $c_1 = c_2$ (8)

store : $(-, ci, t_1, c_2, (c_3, t_3))$ (9)
$\to (-, (c_3, t_1))$ if $c_2 = c_3 \wedge ci \notin \{t_{mallocinit}, t_{freeinit}\}$

store : $(-, t_{mallocinit}, t_1, c_2, F) \to (-, (c_2, t_1))$ (10)

store : $(-, t_{freeinit}, t_1, c_2, (c_3, t_4)) \to (-, F)$ (11)

move : $(-, t_{malloc}, t, -, -) \xrightarrow{1} (-, t_{newtag})$ (12)

move : $(-, \overline{t_{malloc}}, t, -, -) \to (-, t)$ (13)

```
primitive_malloc=malloc;
malloc (int size) {
   void *p=primitive_malloc(size); // orig ptr
   void *tp;  // tagged ptr
   void *tmp; // tagged ptr to individual words
   asm: malloc move r1=p, r2=tp // alloc fresh tag
   tmp=tp;
   for (int i=0;i<size;i++) {
         // set region tag on word in new region
         asm mallocinit store r1=0,r2=tmp
         tmp++;
      }
    return(tp);
}

primitive_free=free;
free (void *p) {
   size =size(p); // size of pointer region
   void *tmp=base(p); // base of pointer region
   for (int i=0;i<size;i++) {
         // set region tag on word in freed region
         asm freeinit store r1=0,r2=tmp
         tmp++;
      }
    return;
}
```

CFI

CFI-1ID [?]

We use 2 tags written as sets: $\varnothing$ and $\{f\}$. The tag $\{f\}$ is used for tagging all indirect control flows, as well as all their potential destinations. The tag $\varnothing$ is used for everything else.

$$\texttt{return}, \texttt{ijump}, \texttt{icall}: \quad (-, \{f\}, -, -, -) \to (\{f\}, -) \tag{1}$$

$$\overline{\texttt{return}, \texttt{ijump}, \texttt{icall}}: \quad (pc, ci, -, -, -) \to (\varnothing, -) \text{ if } pc \subseteq ci \tag{2}$$

CFI-2ID [?]

In this policy r is used for marking returns and their potential targets, and c is used for indirect calls and jumps and their potential targets. Since these two cases can overlap, we're using 4 tags written as sets: $\varnothing$, $\{r\}$, $\{c\}$, and $\{r, c\}$.

$$\texttt{return}: \quad (pc, ci, -, -, -) \to (\{r\}, -) \text{ if } r \in ci, pc \subseteq ci \tag{1}$$

$$\texttt{ijump}, \texttt{icall}: \quad (pc, ci, -, -, -) \to (\{c\}, -) \text{ if } c \in ci, pc \subseteq ci \tag{2}$$

$$\overline{\texttt{return}, \texttt{ijump}, \texttt{icall}}: \quad (pc, ci, -, -, -) \to (\varnothing, -) \text{ if } pc \subseteq ci \tag{3}$$

CCFIR [?]

r is the return-id, c is the call-id, p is the return-into-privileged-code-id. Assuming 6 tags written as the sets: $\varnothing$, $\{r\}$, $\{p\}$, $\{c\}$, $\{r, c\}$, and $\{p, c\}$.

$$\texttt{return}: \quad (pc, ci, -, -, -) \to (\{r\}, -) \text{ if } r \in ci, pc \subseteq ci \tag{1}$$

$$\texttt{return}: \quad (pc, ci, -, -, -) \to (\{p\}, -) \text{ if } p \in ci, pc \subseteq ci \tag{2}$$

$$\texttt{ijump}, \texttt{icall}: \quad (pc, ci, -, -, -) \to (\{c\}, -) \text{ if } c \in ci, pc \subseteq ci \tag{3}$$

$$\overline{\texttt{return}, \texttt{ijump}, \texttt{icall}}: \quad (pc, ci, -, -, -) \to (\varnothing, -) \text{ if } pc \subseteq ci \tag{4}$$

CFI-ROP

We are assuming an allowed control-flow graph χ, containing pairs of a return ID and a possible destination ID. We write IDs as ci or pc below. Tags are either valid IDs or $\perp$.

$$\texttt{return}: \quad (\perp, ci, -, -, -) \to (ci, -) \tag{1'}$$

$$\overline{\texttt{return}}: \quad (pc, ci, -, -, -) \to (\perp, -) \text{ if } (pc, ci) \in \chi \tag{2'}$$

$$\overline{\texttt{return}}: \quad (\perp, -, -, -, -) \to (\perp, -) \tag{3'}$$

$$\texttt{return}: \quad (pc, ci, -, -, -) \to (ci, -) \text{ if } (pc, ci) \in \chi \tag{4'}$$

CFI-JOP

Assuming an allowed control-flow graph, χ.

$$\texttt{ijump}, \texttt{icall}: \quad (\perp, ci, -, -, -) \to (ci, -) \tag{1}$$

$$\texttt{ijump}, \texttt{icall}: \quad (pc, ci, -, -, -) \to (ci, -) \text{ if } (pc, ci) \in \chi \tag{2}$$

$$\overline{\texttt{ijump}, \texttt{icall}}: \quad (\perp, -, -, -, -) \to (\perp, -) \tag{3}$$

$$\overline{\texttt{ijump}, \texttt{icall}}: \quad (pc, ci, -, -, -) \to (\perp, -) \text{ if } (pc, ci) \in \chi \tag{4}$$

Complete-CFI

We assume an allowed control-flow graph χ.

$$\texttt{return}, \texttt{ijump}, \texttt{icall}: \quad (\perp, ci, -, -, -) \to (ci, -) \tag{1}$$

$$\texttt{return}, \texttt{ijump}, \texttt{icall}: \quad (pc, ci, -, -, -) \to (ci, -) \text{ if } (pc, ci) \in \chi \tag{2}$$

$$\overline{\texttt{return}, \texttt{ijump}, \texttt{icall}}: \quad (\perp, -, -, -, -) \to (\perp, -) \tag{3}$$

$$\overline{\texttt{return}, \texttt{ijump}, \texttt{icall}}: \quad (pc, ci, -, -, -) \to (\perp, -) \text{ if } (pc, ci) \in \chi \tag{4}$$

Taint Tracking

$$\texttt{nop}, \texttt{cbranch}, \texttt{ubranch}, \texttt{ijump}, \texttt{return}: \quad (-, -, -, -, -) \to (-, -) \tag{1}$$

$$\texttt{ar2s1d}: \quad (-, ci, op_1, op_2, -) \to (-, ci \cup op_1 \cup op_2) \tag{2}$$

$$\texttt{ar1s1d}: \quad (-, ci, op_1, -, -) \to (-, ci \cup op_1) \tag{3}$$

$$\texttt{ar1d}, \texttt{dcall}, \texttt{icall}, \texttt{flags}: \quad (-, ci, -, -, -) \to (-, ci) \tag{4}$$

$$\texttt{load}: \quad (-, ci, op_1, -, mr) \to (-, ci \cup op_1 \cup mr) \tag{5}$$

$$\texttt{store}: \quad (-, ci, op_1, op_2, -) \to (-, ci \cup op_1 \cup op_2) \tag{6}$$

$$\texttt{move}: \quad (-, t_{taint}, -, -, -) \xrightarrow{1} (-, t_{newtag}) \tag{7}$$

$$\texttt{move}: \quad (-, ci \neq t_{taint}, op_1, -, -) \to (-, ci \cup op_1) \tag{8}$$

Subword operations

The rules above, which we used in our experiments, do not account for subword operations. To properly support subword operation we would need to break up the `load` and `store` opgroups into two opgroups for word operations (`wload` and `wstore`) and two opgroups byte operations (`bload` and `bstore`).

The rules for all policies which explicitly talk about loads or stores would need to change (simple types, memory safety, and taint tracking). Here is how the (no retaddr variant of the) simple types policy would change (the `w` opgroups correspond to the previous rules):

$$\texttt{wload}: \quad (-, \texttt{insn}, \texttt{addr}, -, \texttt{other}) \to (-, \texttt{other}) \tag{1}$$

$$\texttt{wload}: \quad (-, \texttt{insn}, \texttt{addr}, -, \texttt{addr}) \to (-, \texttt{addr}) \tag{2}$$

$$\texttt{wstore}: \quad (-, \texttt{insn}, \texttt{other}, \texttt{addr}, -) \to (-, \texttt{other}) \tag{3}$$

$$\texttt{wstore}: \quad (-, \texttt{insn}, \texttt{addr}, \texttt{addr}, -) \to (-, \texttt{addr}) \tag{4}$$

$$\texttt{bload}: \quad (-, \texttt{insn}, \texttt{addr}, -, \texttt{other}) \to (-, \texttt{other}) \tag{5}$$

$$\texttt{bload}: \quad (-, \texttt{insn}, \texttt{addr}, -, \texttt{addr}) \to (-, \texttt{other}) \tag{6}$$

$$\texttt{bstore}: \quad (-, \texttt{insn}, \texttt{other}, \texttt{addr}, -) \to (-, \texttt{other}) \tag{7}$$

$$\texttt{bstore}: \quad (-, \texttt{insn}, \texttt{addr}, \texttt{addr}, -) \to (-, \texttt{other}) \tag{8}$$

Here are the `b` rules for memory safety:

$$\texttt{bload}: \quad (-, -, c_1, -, (c_2, c_3^{\perp})) \to (-, \perp) \text{ if } c_1 = c_2 \tag{1}$$

$$\texttt{bstore}: \quad (-, ci, c_1^{\perp}, c_2, (c_3, c_4^{\perp})) \to (-, (c_3, \perp)) \text{ if } c_2 = c_3 \wedge ci \notin \{t_{mallocinit}, t_{freeinit}\} \tag{2}$$

Here is the `bstore` rule for taint tracking:

$$\texttt{bstore}: \quad (-, ci, op_1, op_2, mr) \to (-, ci \cup op_1 \cup op_2 \cup mr) \tag{1}$$