An untrusted verifier for Typed Assembly Language

Adam Chlipala

Basic idea

- Typed Assembly Language (TAL) adds a type system over an architecture's assembly language, allowing type safety proofs for a useful subset of machine code programs.
- The Open Verifier is a toolkit for abstract interpretation of machine code programs to prove memory safety.
- An untrusted plugin for a particular compiler describes a verification state in first-order logic for each reachable program location.
- The trusted core of the Open Verifier performs the abstract interpretation, asking the plugin to prove entailments between states, based on a trusted model of machine semantics.
- When a potentially unsafe instruction is encountered, the plugin is asked to prove its safety using only the first-order state it had declared for that location.
- "Higher-order" reasoning can be replaced by logical states that don't specify their program counters exactly.

Example program

In Popcorn (a safe C dialect):

```
struct int_pair { int n1, n2; }

void incrementFirst(int_pair p) {
  p.n1 = p.n1 + 1;
}
```

In TAL:

type `int_pair` $\equiv \textit{word} \times \textit{word}$

`incrementFirst` $: \forall\alpha.\{\mathrm{ESP} : \mathtt{sptr}(\{\mathrm{ESP} : \mathtt{sptr}(\alpha)\} :: \mathtt{int\_pair} :: \alpha)\}$

```
MOV EAX, [ESP+4]
MOV EBX, [unroll(EAX)+0] ;  an unroll coercion expands a named type definition
ADD EBX, 1
MOV [unroll(EAX)+0], EBX
RETN 4 ;  RETN's argument indicates how many extra bytes of the stack to pop off
```

Local state as first-order logic

Logical state at the entry to `incrementFirst`:

$$\exists \Gamma .\ \Gamma \vdash \mathtt{MEM}$$

$$\wedge\ \Gamma \vdash \mathtt{int\_pair} \equiv \mathit{word} \times \mathit{word}$$
$$\wedge\ \Gamma \vdash \mathtt{ESP} : \mathtt{sptr}(\{\mathtt{ESP} : \mathtt{sptr}(\alpha)\} :: \mathtt{int\_pair} :: \alpha)$$

What it means: There exists typing context Γ such that:

- Every allocated cell of the current memory contains a value of the type dictated by Γ.
- Γ's definition of `int_pair` matches the program's.
- The current value of `ESP` is a pointer to a stack that begins with a code pointer, followed by an `int_pair` and a stack tail of the unknown type α. The code pointer points to code that is safe if called when `ESP` points to a stack of type α.

Universal quantification over α is used to force the function to leave the tail of the stack untouched.

Global state as first-order logic

- Logical state to stand for any safe jump:

$$\exists \Gamma, v_1, .., .v_n.\ \Gamma \vdash \mathtt{MEM}$$
$$\wedge\ r_1 = v_1 \wedge ... \wedge r_n = v_n$$
$$\wedge\ \Gamma \vdash \mathtt{int\_pair} \equiv \mathit{word} \times \mathit{word}$$
$$\wedge\ \Gamma \vdash \mathtt{PC} : \{r_1 : \tau_1, ..., r_n : \tau_n\}$$
$$\wedge\ \Gamma \vdash \{r_1 = v_1, ..., r_n = v_n\} : \{r_1 : \tau_1, ..., r_n : \tau_n\}$$

What it means:

- The program counter has a code type that is safe to jump to if each register r_i has type τ_i.
- The current values of the registers do have these types.

With a simple syntactic definition of what it means to have code type, we can prove that any state satisfying this formula is one of the states already queued to be verified.

Proof obligations

When the trusted core reaches this instruction:

$$\texttt{MOV EBX, [}\mathit{unroll}(\texttt{EAX})\texttt{+0]}$$

The state might contain:

$$\Gamma \vdash \texttt{MEM} \wedge \Gamma \vdash \texttt{EAX} : \texttt{int\_pair}$$

The Open Verifier requires a proof that EAX is safe to dereference:

$$\cfrac{\cfrac{\Gamma \vdash \texttt{int\_pair} \equiv word \times word \quad \Gamma \vdash e : \texttt{int\_pair}}{\Gamma \vdash e : word \times word}\ unroll}{safeptr(e)}\ prod\_safe$$

The target state will contain a new predicate $\Gamma \vdash \texttt{EBX} : word$, which must be proved like:

$$\cfrac{\Gamma \vdash m \quad \cfrac{\Gamma \vdash \texttt{int\_pair} \equiv word \times word \quad \Gamma \vdash e : \texttt{int\_pair}}{\Gamma \vdash e : word \times word}\ unroll}{\Gamma \vdash m[e] : word}\ prod\_read\_1$$