A Fully-Abstract Translation of Pointers to Capabilities

Akram El-Korashy, Stelios Tsampas, Marco Patrignani, Dominique Devriese, Deepak Garg, Frank Piessens
MPI-SWS, KU-Leuven, CISPA-Saarland & Stanford
A capability is an access token that gives permission to perform some operation on some resource.
What is a capability-based computer?
What is a capability computer?

Example of a memory instruction:

```
load r1, c4
```

For a successful load, c4 needs to contain a valid "load capability" on a region containing 0xB1005..E.
Role of compiler and loader in a capability computer

- The **loader** places a capability (e.g., in c0) that authorizes memory operations on all of the program’s data segment.

- The **compiler** creates sub-capabilities as needed and uses them for the corresponding instructions.
Our model of a “pointers-to-capabilities” translation
Example program and its translation

```c
#include "networking.h"

iobuffer [512];
static secret;
main() {
    iobuffer[42] = 4242;
    send_rcv(&iobuffer);
    handle_secret();
}
handle_secret() { ... }
```

```c
#include "networking.h"
data_segment_size: 513

main() {
    inc(ddc, 42) = 4242;
    send_rcv(lim(ddc, 0, 512));
    handle_secret();
}
handle_secret() { ... }
```

Spatially-safe semantics for pointers

Secure implementation of this semantics
What is a fully-abstract translation?
What is a fully-abstract translation?

∀P₁ P₂

∀Cₛ. Cₛ[P₁] ≈ Cₛ[P₂] ⇐⇒


The ⇒ direction is called preservation of contextual equivalence

P₁, P₂: Two source programs
P₁↓, P₂↓: Translated versions of P₁ and P₂
Cₛ: Source context
Cₜ: Target context
How to prove the \( \Rightarrow \) direction? Use trace equivalence (1)

\[
\forall P_1, P_2 \\
\forall C_s \cdot C_s[P_1] \approx C_s[P_2] \\
\Rightarrow \\
\forall C_T \cdot C_T[P_1] \downarrow \approx C_T[P_2] \downarrow
\]

Design traces that record all “observable behavior”. Then instead prove:

\[
\forall P_1, P_2 \\
\text{traces}(P_1) = \text{traces}(P_2) \\
\Rightarrow \\
\text{traces}(P_1 \downarrow) = \text{traces}(P_2 \downarrow)
\]

The traces of a program are all its possible interactions with any context.
How to prove the $\Rightarrow$ direction? Use trace equivalence (2)

$\forall P_1, P_2$

\[ \text{traces}(P_1) = \text{traces}(P_2) \]

$\Rightarrow$

\[ \text{traces}(P_1 \downarrow) = \text{traces}(P_2 \downarrow) \]

Show that for any program, its “traces” set does not change after translation

$\forall P. \text{traces}(P) = \text{traces}(P \downarrow)$

i.e., need to show

$\forall P \ t. \ t \in \text{traces}(P) \iff t \in \text{traces}(P \downarrow)$


Ternary simulation to prove the theorem:

$$\forall P \ t. \ t \in \text{traces}(P) \iff t \in \text{traces}(P\downarrow)$$
Example to explain the use of a ternary cross-language simulation relation

\[ \text{C}_{\text{emulating}}[\text{P}] \]

\[ \text{C}_{\text{emulating}}[\text{P}] \approx \text{Lock-step simulation} \]

\[ \text{C}_{\text{emulating}}[\text{P}] \approx \text{Lock-step simulation} \]

\[ \text{send} \_\text{rcv}([\ldots,4242,\ldots]) \]

\[ \text{return}([0,\ldots,0]) \]

\[ \text{send} \_\text{rcv}([\ldots,4242,\ldots]) \]

\[ \text{return}([0,\ldots,0]) \]

\[ \text{C}_{\text{obtained}}[\text{P}] \]

\[ \text{Option simulation} \]

\[ \text{Forward simulation} \]

\[ \text{Emulation invariants} \]

\[ \text{Strengthening} \]
Conclusion:

- Model a pointers-to-capabilities translation.
- Prove that it is fully abstract by using a ternary cross-language simulation for the proof of:

\[ \forall P \ t. \ t \in \text{traces}(P) \Leftrightarrow t \in \text{traces}(P\downarrow). \]

Thank you