Compiling a secure variant of C to capabilities

Akram El-Korashy*, Stelios Tsampas^, Marco Patrignani~, Dominique Devriese^, Deepak Garg*, Frank Piessens^  
*MPI-SWS, ^KU-Leuven, ~CISPA

Dagstuhl Seminar on Secure Compilation
How to keep C variables in RAM in a secure way?

Your intentions may be noble, but they are also misguided. The short answer is that there's really no way to do what you want on a general purpose system (i.e. commodity processors/motherboard and general-purpose O/S).

Data isolation is needed to be able to reason about security invariants.

C semantics does not require any isolation guarantee.
Data isolation?

We mean private state.

Only specific functions should be given access to specific pieces of data.

A programmer may rely on isolation to reason about security while still using untrusted libraries.
Example I

Private state in a C program

Safe fine-grained sharing while still maintaining the privacy of access to \( b \):

```c
int a;
int b;  //private variable

int f() {
    context_f(&a);
    ...
}
```
Example II

Private state in a C program, control flow violation

int secret;

// check then write secret
int check_and_write(t* f) {
    if (check(f)) {
        write(f, secret);
    }
}

---
Approach

1. C-like language with module isolation

2. Compile this language to a capability machine model.

3. Prove that this translation is secure (fully abstract).
A secure variant of C or a restricted subset of C

Modules as units of isolation

Functions within a module can access its global variables.

Jumping into the middle of functions is prohibited by design.

A variant of Clight, with regards to how we represent spatial memory safety
Example I revisited

- Compile pointers as capabilities.
- Sharing a pointer corresponds to sharing a restricted capability.

Safe fine-grained sharing while still maintaining the privacy of access to `b`:

```c
int a;
int b;  //private variable
int f() {
    context_f(&a);
    ...
    ____
```
Example II revisited

● CHERI also features code capabilities.
● But it offers more..

● Sealed capabilities authorize access to code and data only by means of a trusted calling mechanism.
● The calling mechanism manages a trusted call stack.

```c
int secret;

// check then write secret
int check_and_write(t* f) {
    if (check(f)) {
        write(f, secret);
    }
}
```
Fully-abstract translation
Color code

Source

Target
Fully-abstract Translation ↓

∀ \(P_1 \neq P_2\)

∀ \(C_S . C_S[P_1] = C_S[P_2]\)

\(\iff\)

∀ \(C_T . C_T[P_1 \downarrow] = C_T[P_2 \downarrow]\)

Two arbitrary programs are equivalent iff their translations are equivalent

\(P_i\): source program

\(P_i \downarrow\): compiled program
Preservation of contextual equivalence

∀ \( P_1 \) \( P_2 \)

∀ \( C_S \cdot C_S[P_1] = C_S[P_2] \)

\[ \rightarrow \]

∀ \( C_T \cdot C_T[P_{i \downarrow}] = C_T[P_{2 \downarrow}] \)

Contextually-equivalent source programs remain so after translation.
Back-translation to prove preservation

∀ $P_1 P_2$

$\exists C_T \cdot C_T[P_1 \downarrow] \neq C_T[P_2 \downarrow]$

$\rightarrow$

$\exists C_S \cdot C_S[P_1] \neq C_S[P_2]$

A distinguishing target context should exist only if there were a source one.
Traces soundness, then back-translation

∀ \( P_1, P_2 \)

\[ \exists C_T \cdot C_T[P_1 \downarrow] \not\approx C_T[P_2 \downarrow] \]

\[ \implies Tr(P_1 \downarrow) \not\approx_{TR} Tr(P_2 \downarrow) \]

\[ \implies \exists C_S \cdot C_S[P_1] \not\approx C_S[P_2] \]

\( P_i \): source program

\( P_i \downarrow \): compiled program
A trace semantics captures the interaction of a component. Trace actions record sandbox switching and the status of shared memory.

Two compiled programs that have equal sets of traces are proved to be contextually equivalent.
Trace label example

"call? f(5), M, alloc"

Actions are calls or returns.

Argument values and shared memory

Allocator status is also observable
Conclusion and Future

- Translate **fully-abstractly** a C-like source language to a target language that abstracts the capability instruction set.

- Source-to-source transformation that **automates the initialization of sandboxes**

- Work on Compositional CompCert -- or a similar infrastructure for fully-abstract compilation proofs?
Thanks!

Questions and comments
References