When Good Components Go Bad

What are the security guarantees of compartmentalization?

Cătălin Hrițcu (Inria Paris)

HOPE Project
Devastating low-level vulnerabilities

• Languages like C/C++ sacrifice security for efficiency
  – type and memory unsafe:
    • e.g. any buffer overflow is catastrophic
  – root cause, working on fix, but it's challenging:
    • efficiency
    • precision
    • scalability
    • backwards compatibility
    • deployment
Compartmentalization = Practical mitigation

• Main idea: break up security-critical applications into mutually distrustful components with clearly specified privileges

• Enforce components can only interact in a safe way:
  – component separation, call-return discipline, ...

• Secure compilation chain:
  – compiler, linker, loader, runtime, system, hardware

• Use efficient enforcement mechanisms:
  – tagged architecture (micro-policies) — software fault isolation (SFI)
  – hardware enclaves (SGX) — capability machines (CHERI)
What are the security guarantees of compartmentalizing compilation?
Source reasoning vs Undefined behavior

• **Source reasoning**
  = We want to reason formally about security with respect to source language semantics

• **Undefined behavior**
  = can't be expressed at all by source language semantics!

• **Many different examples in a usual C compiler**
  – out of bounds array accesses
  – use after frees and double frees
  – invalid unchecked casts
  – (often even) signed integer overflows,
  – ...
Restricting undefined behavior

- **Mutually-distrustful components**
  - restrict *spatial* scope of undefined behavior
  - *each component protected from all the others*, in particular from compromised components

- **Dynamic compromise**
  - restrict *temporal* scope of undefined behavior
  - *each component gets guarantees* as long as it has not encountered undefined behavior
  - the mere existence of vulnerabilities doesn't immediately make a component compromised
∃ a dynamic compromise scenario explaining $t$ in source language for instance leading to the following compromise sequence:

(0) $C_0 \rightarrow C_1 \rightarrow C_2 \rightsquigarrow ^* m_1;\text{Undef}(C_1)$

(1) $\exists A_1. C_0 \rightarrow A_1 \rightarrow C_2 \rightsquigarrow ^* m_2;\text{Undef}(C_2)$

(2) $\exists A_2. C_0 \rightarrow A_1 \rightarrow A_2 \rightsquigarrow t$

When Good Components Go Bad (arXiv:1802.00588)
Building secure compilation chain

(mostly) Verified in Coq

Compartmentalized unsafe source

Buffers, procedures, components interacting via strictly enforced interfaces

Compartmentalized abstract machine

Simple RISC abstract machine with build-in compartmentalization

Micro-policy machine

Software fault isolation

Bare-bone machine

Systematically tested (with QuickChick)

When Good Components Go Bad (arXiv:1802.00588)