Towards Formal Verification of Memory Management Properties using Separation Logic

Nicolas Marti
University of Tokyo

Reynald Affeldt
AIST-RCIS

Akinori Yonezawa
University of Tokyo
AIST-RCIS
Research Project

• Verification of low-level software:
  – Specialized operating systems
  – Device drivers

• Difficulties:
  – Memory management
  – Hardware-dependent specifications

• Our approach:
  – Verification in the Coq proof assistant [INRIA, 1984-2005]
  – Using Separation Logic [Reynolds et al., 1999-2005]
This Presentation

• Use-case:
  – The Topsy operating system:
    • Specialized o.s. for network cards [Ruf, ANTA 2003]
    • Also used for educational purpose (in Swiss)
  – Verification of memory isolation:
    • Intuitively, “user-level threads cannot access kernel-level memory” [Bevier, IEEE Trans. 1988]
    • Obvious relation with security:
      – E.g., a user application replacing the process descriptor of an authentication server

• Coq implementation overview
Outline

• Memory Isolation for Topsy
  – Specification Approach
  – Informal Specification

• Excerpt of Formal Verification
  – The Allocation Function
  – Formal Specification and Verification

• Coq Implementation

• Related and Future Work
Memory Isolation for Topsy

• Reminder:
  – “user-level threads cannot access kernel-level memory”

• In practice (for x86 processors):
  – Each thread and segment is given a privilege level
  – The hardware guarantees that user-level threads can only access user-level segments…
    …under the hypothesis that the operating system correctly manages privilege levels!
Where do We Need to Look?

- Topsy control-flow:
What do We Need to Verify?

• Topsy source code:
Memory Isolation for Topsy

• Informal specification:
  – Boot loader and kernel initialization:
    • The boot loader builds the intended memory model and the processor runs in segmented mode
  – Heap manager:
    • Newly allocated blocks do not override previously allocated blocks and only free blocks are marked as such
  – Thread manager:
    • Thread descriptors for user-level threads are initialized with user privilege and context switching preserves this privilege

See paper and website for details

Next slides
Outline

• Memory Isolation for Topsy
  – Specification Approach
  – Informal Specification

 Excerpt of Formal Verification
  – The Allocation Function
  – Formal Specification and Verification

• Coq Implementation

• Related and Future Work
The Allocation Function

• Signature:

\[
\text{hmAlloc (y, sizey);}
\]

• The underlying data structure:
  – Blocks organized as a list
    • E.g., a heap-list with two free blocks and one allocated block:

  – The “heap-list” covers a fixed region of memory reserved by the kernel
hmAlloc: Implementation

- Overall effect:

```c
if (y == 0)
    return ERROR;
/* split the found block to the appropriate size */
split (y, sizey);
/* if not found, compact and search again */
if (y == 0) {
    compact();
y = findFree (sizey);
}
if (y == 0)
    return ERROR;
/* split the found block to the appropriate size */
split (y, sizey);
```
Potential Problems Relevant to Memory Isolation

• Unexpected situations:

⇒ Separation logic [Reynolds et al., 1999-2005] provides convenient formulas for such specifications
Separation Logic Formulas

• Provides a symbolic representation of memory storage:
  – Atoms:
    • E.g., \((l_0 \mapsto e_0)\)
  – Separating conjunction:
    • P \(*\) Q holds when the storage can be split into two parts that respectively satisfy P and Q
    • E.g., \((l_0 \mapsto e_0) * (l_1 \mapsto e_1)\) does not hold if \(l_0=l_1\)
  – Neutral: emp
The Heap-list Predicate

• The Array predicate:
  – An array is a set of contiguous locations

• The Heap-list predicate:
  – Inductively, a heap-list is either:
    • An empty list, or
    • A free block followed by a heap-list, or
    • An allocated block followed by a heap-list

Formal predicates:

Array \( l \ sz = \)

\( (sz = 0 \land \text{emp}) \lor \)

\( (sz > 0 \land ((\exists e.l \mapsto e) \land (\text{Array} \ (l + 1) \ (sz - 1)))) \)

Heap-list \( l = \)

\( \exists st.(l \mapsto st, \text{nil}) \lor \)

\( \exists next.(next \neq \text{nil}) \land (l \mapsto \text{free}, next) \land \)

\( (\text{Array} \ (l + 2) \ (next - l - 2)) \land (\text{Heap-list} \ next) \lor \)

\( \exists next.(next \neq \text{nil}) \land (l \mapsto \text{allocated}, next) \land \)

\( (\text{Array} \ (l + 2) \ (next - l - 2)) \land (\text{Heap-list} \ next) \)
The Heap-List Predicate (cont’d)

- Heap-lists “with holes”:
  - Heap-List $AFx$ holds for a heap-list without the blocks in $A$ or $F$
  - E.g.:
    - Heap-List $\{\} \{\} x$ holds for
    - Heap-List $\{\} \{f\} x$ holds for
    - Heap-List $\{a\} \{\} x$ holds for
Formal Specification of hmAlloc

\[
\begin{aligned}
\{ & \text{Heap - List } \{ x \} \{ y \} \text{ hm } _{\text{base}} * \text{ Array } x \text{ size } x \\
& \text{hmAlloc } (y, \text{size } y); \\
& \exists \text{size } x \geq \text{size } y \land \\
& \text{Heap - List } \{ x, y \} \{ y \} \text{ hm } _{\text{base}} * \text{ Array } x \text{ size } x * \text{ Array } y \text{ size } \\
& \lor \\
& y = 0 \land \text{Heap - List } \{ x \} \{ y \} \text{ hm } _{\text{base}} * \text{ Array } x \text{ size } x
\end{aligned}
\]
Proof Overview (1/2)

\[
\{\text{Heap-List } \{x\}\{y\} hm\_base * \text{Array } x \text{ size}_x \}
\]

\[
y = \text{findFree (size}_y); \\
\text{if } (y == 0) \{
\text{compact ();} \\
y = \text{findFree (size}_y); \\
\}
\]

\[
\exists \text{size.size} \geq \text{size}_y \land \\
\text{Heap-List } \{x\}\{y\} hm\_base * \text{Array } x \text{ size}_x * \text{Array } y \text{ size} \\
\lor \\
y = 0 \land \text{Heap-List } \{x\}\{} hm\_base * \text{Array } x \text{ size}_x
\]
Proof Overview (2/2)

\[
\begin{align*}
&\exists size.size \geq size_y \land \\
&\text{Heap-List } \{x\} \{y\} hm\_base \ast \text{Array } x \text{ size}_x \ast \text{Array } y \text{ size} \\
&\quad \lor \\
&\quad y = 0 \land \text{Heap-List } \{x\} \{\} hm\_base \ast \text{Array } x \text{ size}_x \\
&\quad \text{if } (y == 0) \{ \\
&\quad \quad \text{return ERROR;} \\
&\quad \} \\
&\quad \text{split } (y, size_y); \\
\end{align*}
\]

\[
\begin{align*}
&\exists size.size \geq size_y \land \\
&\text{Heap-List } \{x, y\} \{\} hm\_base \ast \text{Array } x \text{ size}_x \ast \text{Array } y \text{ size} \\
&\quad \lor \\
&\quad y = 0 \land \text{Heap-List } \{x\} \{\} hm\_base \ast \text{Array } x \text{ size}_x \\
\end{align*}
\]
Outline

• Memory Isolation for Topsy
  – Specification Approach
  – Informal Specification

• Excerpt of Formal Verification
  – The Allocation Function
  – Formal Specification and Verification

  Coq Implementation

• Related and Future Work
Coq Implementation

• **Reusable part** (around 6500 lines):
  – Core separation logic
    [Reynolds, LICS 2002]
  – Additional facilities
    • Data structures, lemmas, etc.

• **Use-case part** (around 4500 lines):
  – Translation of Topsy functions
    • C and assembly code (around 300 lines)
  – Specification and verification
    • In progress (some elementary steps left out for lack of time)
# Coq Implementation

## (Reusable part)

<table>
<thead>
<tr>
<th>Core separation logic</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational semantics</td>
<td>Soundness</td>
</tr>
<tr>
<td>$st \xrightarrow{c} st'$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>States</th>
<th>Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>=</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Additional facilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data structures (arrays, lists), lemmas (split, concatenation, insertion, etc.), weakest preconditions generator (triples for backward reasoning), frame rule (for compositional reasoning), tactics for heap partitions</td>
</tr>
</tbody>
</table>

Assignments, pointer dereferences, destructive updates, loops, etc.

Hoare triples

$\{P\}c\{Q\}$
Outline

• Memory Isolation for Topsy
  – Specification Approach
  – Informal Specification

• Excerpt of Formal Verification
  – The Allocation Function
  – Formal Specification and Verification

• Coq Implementation

Related and Future Work
Related Work

• Proof assistant-based verification:
  – Verification of micro-kernels:
    • Delta-core [Zhu et al., O.S.Review 2001]
      – Commercial o.s. verified in PowerEpsilon
      – Verification of error-recovery of system calls
    • VFiasco [Hohmuth and Tews, ECOOP-PLOS 2005]
      – C++ translation into PVS
  – Verification of C programs:
    • Schorr-Waite algorithm in Coq [Hubert and Marche, SEFM 2005]
  – Separation logic encoding:
    • In Isabelle [Weber, CSL 2004]

• Verification using separation logic:
  – Decidable fragment [Berdine et al., FSTTCS 2004]
  – Symbolic evaluator [Berdine et al., APLAS 2005]
Future Work

• Implementation in progress:
  – Complete libraries of lemmas for data structures
  – Polish verification of memory isolation for Topsy

• Automate verification:
  – Interface with the symbolic evaluator of [Berdine et al., APLAS 2005]:
    • Verification of their implementation as a side-effect
  – Semi-automatic generation of loop invariants
  – Interface with theorem provers for BI logic?
Conclusion

• We have presented:
  – A reusable implementation of separation logic in the Coq proof assistant
  – A real-world use-case: memory isolation for the Topsy operating system
    • Overview of memory allocation, see the paper and the website for the rest of the verification (boot loader, memory and thread management)