ProvenCore: Towards a Verified Isolation Micro-Kernel

Stéphane Lescuyer

Prove & Run

MILS Workshop, Amsterdam, 20/01/2015
Summary

1. ProvenCore
   - Overview
   - Features
   - Policies

2. ProvenTools
   - The Smart language
   - Tool-chain
   - C code generation

3. Proofs and properties
   - Refinements
   - Properties
ProvenCore’s objectives

Objectives

- formally prove security properties of a $\mu$-kernel
  - absence of runtime-errors
  - functional specifications
  - integrity / confidentiality
- using IDE and language developed at Prove & Run
- target high-level Common Criteria evaluation
- generate documentation from the specs
- generate executable C code from the specs
ProvenCore’s objectives

Objectives

- formally prove security properties of a $\mu$-kernel
  - absence of runtime-errors
  - functional specifications
  - integrity / confidentiality
- using IDE and language developed at Prove & Run
- target high-level Common Criteria evaluation
- generate documentation from the specs
- generate executable C code from the specs
  - take advantage of the proof effort
Typical mobile device architecture
Typical mobile device architecture

- no assets are safe
- *Corporate Owned, Personally Enabled*
Securing the rich kernel?

- theoretically possible...
- ...but in practice too complex, moving target
TrustZone to the rescue
TrustZone to the rescue

→ the OS on the secure side can be constrained!
ProvenCore as a secure world OS

Overview

ProvenCore as a secure world OS

- **Normal World**: Applications
  - Operating System
  - ProvenCore

- **Secure World**: Applications

- **Monitor World**: TrustZone Monitor
  - TrustZone

- **Hardware**: Trusted Computing Base

- **Software**:
  - Possible safe BYOD policy
  - Possible to run a TEE on ProvenCore
ProvenCore as a secure world OS

- possible safe BYOD policy
- possible to run a TEE on ProvenCore

ProvenCore as a secure world OS
ProvenCore is largely inspired by Minix 3.1
ProvenCore is largely inspired by Minix 3.1

Why Minix?

- $\mu$-kernel
- well-documented
- mostly POSIX-compliant
- simple yet versatile enough
  - no real-time constraints
Minix heritage

ProvenCore is largely inspired by Minix 3.1

Why Minix?
- \( \mu \)-kernel
- well-documented
- mostly POSIX-compliant
- simple yet versatile enough
  - no real-time constraints

Still...
- we had to port it to ARM and MMU
- not a well-defined TCB
The TCB incident

INIT

SHELL

...  

PM

FS

...  

SYSTEM

KERNEL

CLOCK

Kernel data structures

PM data

IPC call

Function call
The TCB incident

- INIT
- SHELL
- ...
- PM
- FS
- ...
- SYSTEM
- KERNEL
- CLOCK
- Kernel data structures

PM data

IPC call
Function call

Stéphane Lescuyer (Prove & Run)
Towards sequentiality

Non-sequential TCB
- 4 different processes, 2 different address spaces
- (blocking) IPC communications inside the TCB
- asynchronous hardware interrupts in the TCB
- behaviour depends on the non-demotion of PM in the scheduler
Towards sequentiality

Non-sequential TCB

- 4 different processes, 2 different address spaces
- (blocking) IPC communications inside the TCB
- asynchronous hardware interrupts in the TCB
- behaviour depends on the non-demotion of PM in the scheduler

→ formal reasoning on a concurrent TCB possible but extremely hard
→ semi-formal reasoning possible, but unsatisfactory
Sequential TCB

Initiator: TCB
Kernel: System
Clock: Fs
PM: PM data
IPC call
Function call

Stéphane Lescuyer (Prove & Run)
Sequential TCB

Stéphane Lescuyer (Prove & Run)
ProvenCore’s features

Process Management
- FORK
- EXIT
- EXEC of authorized codes
ProvenCore’s features

Process Management

- FORK
- EXIT
- EXEC of authorized codes

IPC

- synchronous message-passing IPCs with timeouts
- asynchronous notifications
ProvenCore’s features

Process Management

- FORK
- EXIT
- EXEC of authorized codes

IPC

- synchronous message-passing IPCs with timeouts
- asynchronous notifications

Data transfers

- process-to-process data copies, guarded by authorizations
- safe shared memory system
Security policies

Resource allocation policy

Access control policy

Information flow policy
Security policies

Resource allocation policy

- **time**: configurable scheduling priority bounds
- **RAM**: physical quotas can be put on binaries

Access control policy

Information flow policy
Security policies

Resource allocation policy

Access control policy

- k-calls configurable bitmaps
- copy precise transferable R/W/RW authorities
- shm exactly one RW access at a time

Information flow policy
Security policies

Resource allocation policy

Access control policy

Information flow policy

IPCs allowed calls configurable
IPCs allowed endpoints configurable
High-level properties

Integrity

Resources (registers, data, code) of a process $P$ can only be modified by another process $Q$ provided $P$ explicitly allowed it, and by the kernel following a request of $P$.

Confidentiality

Resources (registers, data, code) of a process $P$ can only be read by another process $Q$ provided $P$ explicitly allowed it, and by the kernel following a request of $P$. 
Prove & Run’s Smart language

**Smart**
- first-order polymorphic functional language
- used *both* for models and specifications
- built-in algebraic datatypes
- **pure**: only manipulate values
Prove & Run’s Smart language

**Smart**
- first-order polymorphic functional language
- used *both* for models and specifications
- built-in algebraic datatypes
- pure: only manipulate values

**Strong separation of data-flow & control-flow**
- each *predicate* returns a number of outputs, as well as *labels*
- labels can typically be used for exceptional cases, or booleans
- only outputs associated to the returned label are created
- ignored labels lead to proof obligations
Smart prototype examples

Labels as exceptions

```java
public get(array<A> a, int idx, A v+) -> [true, oob]
program { ... }
```
Smart prototype examples

Labels as exceptions

```java
public get(array<A> a, int idx, A v+) -> [true, oob]
program { ... }
```

Pure control-flow predicates

```java
public is_slot_free(proc p) -> [true, false]
program { ... }
```
Smart prototype examples

Labels as exceptions

```java
public get(array<A> a, int idx, A v+) -> [true, oob]
program { ... }
```

Pure control-flow predicates

```java
public is_slot_free(proc p) -> [true, false]
program { ... }
```

No side-effect hidden

```java
public fetch_irq(gic gic0, nat irq+, gic gic+)
implicit program
```
Tool-chain overview

Code + Specs

Smart

Smil

Documentation
Proof Obligations
C Source Code
Screenshots
The abstract type of efficient contiguous arrays containing elements of type \(\text{N}\). The type \(\text{N}\) is used to represent the size shared by all arrays in the type, in a fashion similar to \text{std::array}. The type \(\text{S}\) is used to represent the arrays’ location in concrete memory, because one instance of \text{vector} is used to denote the various possible values of a single array in memory. Therefore, values of this type can only be used in a \text{global} and \text{linear} fashion. Consequently, we do not provide a way to create such objects from scratch either.

An extra limitation is that this type is not suitable for arrays of size 0.

- \text{index}
- \text{get}

Returns the element at index \(\text{idx}\) in \(\text{a}\).

Cannot fail because all indices in the type \text{index}\(\text{S}, \text{N}, \text{A}\) are valid for vectors of type \text{vector}\(\text{S}, \text{N}, \text{A}\).

- \text{set}

Sets the element at index \(\text{idx}\) in \(\text{a}\) to \(\text{value}\), and returns the new \text{vector} \(\text{b}\).

Cannot fail because all indices in the type \text{index}\(\text{S}, \text{N}, \text{A}\) are valid for vectors of type \text{vector}\(\text{S}, \text{N}, \text{A}\).

- \text{begin}
- \text{next}

Given an \text{index} into a \text{vector} of type \text{vector}\(\text{S}, \text{N}, \text{A}\), returns the index of the next element in the \text{vector}.

Raisess **exit** if \(\text{idx}\) was the last element of the \text{vector}.

- \text{last}
- \text{length}
- \text{ordinal}
- \text{index}
- \text{lc_index}
Refinements

- SPM
- RSM
- FSP
- TDS
Refinements

- SPM
- RSM
- FSP
- TDS

TDS  Target Design
Refinements

SPM  RSM  FSP  TDS

SPM Security Policy Model
RSM Refined Security Model
FSP Functional Specifications
TDS Target Design
Refinements

SPM

RSM

FSP

TDS

RSM  Refined Security Model

FSP  Functional Specifications

TDS  Target Design
Refinements

- **SPM**  Security Policy Model
- **RSM**  Refined Security Model
- **FSP**  Functional Specifications
- **TDS**  Target Design
Forward Simulation

$\phi$ a view from concrete states to abstract states

\[
\begin{array}{c}
  a_0 \rightarrow a_1 \\
  \phi \uparrow \quad \phi \uparrow \\
  c_0 \rightarrow c_1
\end{array}
\]
Forward Simulation

$\phi$ a view from concrete states to abstract states

$\phi$ $\uparrow$

$\phi$ $\uparrow$

$c_0 \rightarrow c_1 \rightarrow \cdots \rightarrow c_n$

$a_0 \rightarrow a_1 \rightarrow \cdots \rightarrow a_n$
Forward Simulation

$\phi$ a view from concrete states to abstract states

\[
\begin{array}{cccc}
\alpha_0 & \sim & \alpha_1 & \sim & \cdots & \sim & \alpha_n \\
\psi & \uparrow & \psi & \uparrow & \psi & \uparrow & \psi \\
\phi & \phi & \phi & \phi & \phi & \phi & \phi \\
\end{array}
\]

- **SPM**
- **RSM**
- **FSP**
- **TDS**
FSP characteristics

- functional code simulating the TDS
- simplified data structures
- simplified algorithms
- linearized address spaces
FSP characteristics

- functional code simulating the TDS
- simplified data structures
- simplified algorithms
- linearized address spaces

→ **Structural** invariants captured
→ functional invariants easier to describe
→ no more bad pointers, overflows, etc
Schematic view of RSM
SPM characteristics

Security model characteristics

- non-deterministic transition system
  - uses *predictions* to simulate external ND
- all processes have their own resources
- generalization of non-critical heuristics
SPM characteristics

Security model characteristics

- non-deterministic transition system
  - uses *predictions* to simulate external ND
- all processes have their own resources
- generalization of non-critical heuristics

→ functional invariants captured
→ processes are isolated by construction
→ proof and expression of security properties
Schematic view of the SPM

CONTROLLER

Machine 1

Regs

Data

Code

Machine 2

Regs

Data

Code

Machine 3

Regs

Data

Code

Machine n

Regs

Data

Code

.....
Conclusion

Prove & Run methodology
- one **unique** language for all levels of abstraction
- **formal** refinements between all levels, in Smart
- automatic **generation** of C code from TDS
- integrated into the Eclipse IDE

ProvenCore
- $\mu$-kernel for secure world on TrustZone aware devices
- formally established integrity and confidentiality
- can host a TEE as a user service
- higher-level models can be reused for other isolation kernels
Appendices
In-place updates, example 1

```c
@Ghost
type ram = ...;

@Global
@Linear
struct glo {
    ...
    ram ram;
    ...
}
```

Stéphane Lescuyer (Prove & Run)
In-place updates, example 1

```plaintext
@Ghost
type ram = ...

@Global
@Linear
struct glo {
    ...
    ram ram;
    ...
}

private enqueue(global now, int rp, global after+)
```
In-place updates, example 1

```c
@Ghost
type ram = ...

@Global
@Linear
struct glo {
    ...
    ram ram;
    ...
}

private enqueue(global now, int rp, global after+)

→ void sm.pred_enqueue(int sm_loc_rp);
```
In-place updates, example 2

```java
@MustAlias(‘‘a=b’’)
public void set(array<int> a, int idx, int v,
               array<int> b+)
    -> [true, out_of_bounds]

... {
    ...
    set(a, i, v, b+);
    set(b, j, w, c+);
    ...
    get(a, j, x+);    // Not OK
    }```

Non-linearity → \(a\) is read after modification of \(a\) in \(b\) → impossible to update \(a\) in-place and alias \(a\) and \(b\) → the analysis produces an error or a copy
In-place updates, example 2

```java
@MustAlias('a=b')
public set(array<int> a, int idx, int v,
        array<int> b+) -> [true, out_of_bounds]

... {
  ...
  set(a, i, v, b+);
  set(b, j, w, c+);
  ...
  get(a, j, x+); // Not OK
}
```

Non-linearity

→ a is read after modification of a in b
→ impossible to update a in-place and alias a and b
→ the analysis produces an error or a copy
In-place updates, example 2

```c
... {
  ... 
  set(a, i, v, b+);  
  set(b, j, w, c+); 
  ... 
  get(c, j, x+);  // OK
}
```
In-place updates, example 2

```c
... {
    ...
    set(a, i, v, b+);
    set(b, j, w, c+);
    ...
    get(c, j, x+); // OK
}
```

```c
set_result_t set(int *a, int idx, int v);

... {
    set(&a, i, v);
    set(&a, j, w);
    ...
    get(&a, j, x+);
}
```
Abstraction example TDS → FSP (1/2)

TDS

```c
struct state {
    array<cell> data;
    option<int> head;
    t current;
};
```

```c
struct cell {
    t value;
    option<int> next;
};
```

Invariants

- all indices valid
- no cycles
Abstraction example TDS $\rightarrow$ FSP (1/2)

```plaintext
TDS

```struct` state {
    array<cell> data;
    option<int> head;
    t current;
}

```struct` cell {
    t value;
    option<int> next;
}

```head: Some 2 ----,
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>v</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td></td>
<td>yy</td>
<td></td>
<td>zz</td>
<td>tt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Some 4</td>
<td></td>
<td>Some 0</td>
<td></td>
<td>Some 5</td>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Invariants
- all indices valid
- no cycles

Stéphane Lescuyer (Prove & Run)
ProvenCore
MILS Workshop, 20/01/15 30 / 25
Abstraction example TDS $\rightarrow$ FSP (1/2)

TDS

```c
struct state {
    array<cell> data;
    option<int> head;
    t current;
}
```

```c
struct cell {
    t value;
    option<int> next;
}
```

Invariants

- all indices valid
- no cycles
Abstraction example TDS → FSP (2/2)

```
FSP

struct state {
    list<t> cells;
    t current;
}

type list<A> =
| Nil
| Cons(A a, list<A> l);
```
Abstraction example TDS → FSP (2/2)

```
FSP

struct state {
    list<t> cells;
    t current;
}

type list<A> =
    | Nil
    | Cons(A a, list<A> l);
```

Properties
the list is sorted
insertion/deletion/... preserve the sortedness
sorted in FSP
⇒ sorted in TDS
Abstraction example TDS → FSP (2/2)

```plaintext
struct state {
    list<t> cells;
    t current;
}

type list<A> =
    | Nil
    | Cons(A a, list<A> l);
```

Properties
- the list is sorted
- insertion/deletion/... preserve the sortedness
- sorted in FSP ⇒ sorted in TDS
Expression of high-level properties

Integrity

Let $s$ be an SPM state, $\bar{\pi}$ some predictions, and $t$ such that $s, \bar{\pi} \rightarrow^* t$, et $i$ the index of a machine in $s$ that never runs during these transitions. Then,

(i) $s_i$ and $t_i$ have identical codes

(ii) $s_i$ and $t_i$ have identical registers unless $i$ is unblocked between $s$ and $t$

(iii) the value at address $p$ in the memory of $s_i$ is the same as in $t_i$, unless $p$ was writable in $s_i$, which happens when:

- $s_i$ is waiting to receive a message and $p$ belongs to the range where the message should be received
- $s_i$ has a write memory permission in its memory that covers $p$

Corollary A machine without write permissions which has been preempted finds its registers, code and data completely unchanged when it is rescheduled.
Expression of high-level properties

Integrity

Let \( s \) be an SPM state, \( \pi \) some predictions, and \( t \) such that \( s, \pi \rightarrow^* t \), et \( i \) the index of a machine in \( s \) that never runs during these transitions. Then,

(i) \( s_i \) and \( t_i \) have identical codes

(ii) \( s_i \) and \( t_i \) have identical registers unless \( i \) is unblocked between \( s \) and \( t \)

(iii) the value at address \( p \) in the memory of \( s_i \) is the same as in \( t_i \), unless \( p \) was writable in \( s_i \), which happens when:

- \( s_i \) is waiting to receive a message and \( p \) belongs to the range where the message should be received
- \( s_i \) has a write memory permission in its memory that covers \( p \)

Corollary

A machine without write permissions which has been preempted finds its registers, code and data completely unchanged when it is rescheduled.
Expression of high-level properties

Similarity

Two SPM states $s$ and $t$ are similar modulo the machine $i$, $s \sim_i t$, if all machines except $i$ are identical between $s$ and $t$. 
Expression of high-level properties

**Similarity**

Two SPM states $s$ and $t$ are similar modulo the machine $i$, $s \sim_i t$, if all machines except $i$ are identical between $s$ and $t$.

**Confidentiality**

Let $s$ and $s'$ be two states similar modulo $i$, and $t$ and $t'$ such that:

- $s, \bar{\pi} \rightarrow^* t$ and $s', \bar{\pi} \rightarrow^* t'$
- $i$ does not run between $s$ et $t$
- $s_i$ has no data readable from another machine, i.e.
  - $s_i$ is not blocked trying to send a message
  - $s_i$ has no read permissions on its memory

Then, $t \sim_i t'$.