Safer Sloth: Efficient Hardware-Tailored Memory Protection

Daniel Danner, Rainer Müller, Wolfgang Schröder-Preikschat, Wanja Hofer, Daniel Lohmann

April 15, 2014
Sloth kernels use hardware for OS purposes, and

- are concise (200–500 LoC)
- are small (300–900 bytes)
- are fast (latency speed-up 2x to 170x)
- implement industry standards (OSEK, OSEKtime, AUTOSAR OS)
**SLOTH Recap**

**Main Idea**

Threads are interrupt handlers, synchronous thread activation is IRQ
⇒ Interrupt subsystem does scheduling and dispatching work

![Diagram of SLOTH's main idea](image-url)
Safer Sloth: Motivation and Goals

Threads as Interrupts?
- But what about safety?

Motivation for Safer Sloth
- Sloth has been criticized for lack of isolation
  - Tasks are executed in IRQ handler context
  - Application code has supervisor privileges

Goals
- Effectively isolate kernel and application
- Maintain design principles of Sloth
Memory Protection in Embedded Systems

- Safety, but not security
- Protect the data, but not the code
- Safety model based on AUTOSAR OS
- MPU-based isolation

- Vertically: Protect kernel state and MPU configuration
- Horizontally: Isolate applications or even tasks from each other
Exploit as much knowledge about target hardware as possible

Tailor kernel to fit both the platform and the application

Taking into account:
- Extent and layout of MPU configuration
- Method for re-programming the MPU
- Available hardware privilege levels
- Is MPU active in all levels?
- Degree of safety required by the application
### Protection Modes in Safer Sloth

**Unsafe**
- The original Sloth OS, without isolation

**MPU**
- MPU active, but tasks execute with supervisor privileges
- Vertical isolation ensured constructively in post-validation

**MPU+traps**
- Vertical isolation ensured by hardware privilege levels
- System services acquire kernel privileges via syscall mechanism
System Configuration:
- Protection mode: unsafe
- MPU
- MPU+traps

Application Configuration:
- Tasks:
  - Task1
  - Task2
  - Task3
  - privlevel: 0
  - privlevel: 1
  - privlevel: 0
- Domains:
  - Dom1
  - Dom2
  - Dom3
- Data:
  - var1
  - var2
  - var3
  - var4

Hardware Model:
- TriCore TC1796:
  - MPU ranges: 4
  - MPU range sets: 2
  - privilege levels: 3
    - 0 = user mode
    - 1 = user mode with periphery access
    - 2 = supervisor mode
  - MPU active in supervisor mode: ✔
  - post-validation available: ✔

Task → MPU Ranges Map
- Syntax: [range idx] = {<from>, <to>}
- Task1 MPU Ranges:
  - [0] = {BEGIN_stack, %sp}
  - [1] = {BEGIN_Dom1, END_Dom1}
- Task2 MPU Ranges:
  - [0] = {BEGIN_stack, %sp}
  - [1] = {BEGIN_Dom2, END_Dom2}
  - [2] = {BEGIN_Dom3, END_Dom3}
- Task3 MPU Ranges:
  - [0] = {BEGIN_stack, %sp}
  - [1] = {BEGIN_Dom3, END_Dom3}

Validation

Generation

Memory Map
- Stack
  - ... → BEGIN_stack → END_stack
- Dom1
  - var1 → END_Dom1
- Dom2
  - var2 → BEGIN_Dom1
- Dom3
  - var3 → var4
  - 0x0

Binary

Post-Validation
**MPU mode in Safer Sloth**

**unsafe Mode:**

```c
Task1:
...
; inlined call to
; GetResource(Res1):
prio = getCurPrio();
pushResourceStack(prio);
if (Res1 > prio) {
    setCurPrio(Res1);
}
...
```

**MPU Mode:**

```c
Task1:
...
; disable MPU
mfcr %d15,$psw
insert %d15,%d15,0,12,1
mtcr $psw,%d15
prio = getCurPrio();
pushResourceStack(prio);
if (Res1 > prio) {
    setCurPrio(Res1);
}
; enable MPU
mfcr %d15,$psw
insert %d15,%d15,15,12,1
mtcr $psw,%d15
...
```

**System service implementation**

- **Supervisor mode**
- **User mode**
- **System service implementation**
MPU+traps mode in Safer Sloth

MPU+traps Mode:

Task1:
...
syscall 2
...

real_GetResource:
prio = getCurPrio();
pushResourceStack(prio);
<...>
<non-inlined implementation>
<...>
rfe

trap_6: ; syscall trap
<syscall dispatcher>
ji ...

System service implementation
Supervisor mode
User mode
generates trap
jumps back
returns to

Daniel Danner
Safer Sloth: Efficient, Hardware-Tailored Memory Protection (April 15, 2014)
The Problem with Traps

- Sloth gains a lot through compiler optimizations. System services are short, parameters are mostly static, compilation done as a single unit.
  - Inlining of system service calls
  - Removal of dead code
  - Constant propagation

Traditional traps prohibit such optimizations

System services must be standalone functions, jumped to via a syscall dispatcher

Solution: Combination of MPU and MPU+traps mode

⇒ Inline traps as 4th protection mode (MPU+itraps)
Inline Traps in **Safer Sloth**

**MPU+itraps Mode:**

```plaintext
Task1:
...
syscall 0
prio = getCurPrio();
pushResourceStack(prio);
if (Res1 > prio) {
  setCurPrio(Res1);
}

; load current pc
mfcr  %d15,$pc
add  %d15,2
; overwrite return address
mov.a  %a11,%d15
rfe
...
```

`generates trap`

`jumps back`

`returns to`

`; syscall trap
trap_6:
    ji  %a11`

**User mode**

**Supervisor mode**

**System service implementation**

Daniel Danner

**Safer Sloth: Efficient, Hardware-Tailored Memory Protection (April 15, 2014)** 12
Evaluation Setup

Evaluation platform: Infineon TriCore TC1796
- 32-bit RISC μ-Controller, clocked at 50 MHz
  - widely used in the automotive industry (BMW, Audi, ...)
  - IRQ system with 256 priority levels and 181 IRQ sources

Safety features:
- 3 privilege levels
- MPU with 2 protections sets, 4 memory ranges each

Comparison against: Commercial AUTOSAR OS
- Offers two modes of protection, equivalent to
  - unsafe mode
  - MPU+traps mode

Approach:
- Microbenchmarks of system service overheads including possible transitions
Evaluation Results: Total Overheads

Safer Sloth vs. commercial AUTOSAR OS

Daniel Danner
Safer Sloth: Efficient, Hardware-Tailored Memory Protection (April 15, 2014)
Evaluation Results: Additional Overheads

Safer Sloth vs. commercial AUTOSAR OS

- Safer Sloth (unsafe mode)
- Δ MPU / +traps / +itraps
- AUTOSAR OS (Δ MPU+traps)

Cycles


Daniel Danner
Safer Sloth: Efficient, Hardware-Tailored Memory Protection (April 15, 2014)
Conclusions

Safer Sloth...

- provides effective memory protection
- offers tailorability to both hardware and application
- maintains advantages of Sloth:
  - no rate-monotonic priority inversions
  - small footprint
  - minimal and constant latencies

⇒ excellent real-time characteristics