A Formal Model of Cache Speculation Side-Channels

Catalin Marinas
Arm Limited

MODULE *CacheSpecv1*

EXTENDS *Sequences*, *FiniteSets*, *TLC*

CONSTANTS

$REGS$,	set of CPU registers ($e.g.$ $\{r1, r2\}$)
$LADDRS$,	set of low memory addresses ($e.g.$ $\{l1, l2\}$)
$HADDRS$,	set of high memory addresses ($e.g.$ $\{h1, h2\}$)
$DATA$,	set of primitive data values ($e.g.$ $\{d1, d2\}$)
$MODE$,	initial security/privilege mode ("low" or "high")
LOG	boolean: CPU instruction trace

ASSUME

$\land\ MODE \in \{\text{"low"}, \text{"high"}\}$
$\land\ LOG \in \text{BOOLEAN}$

$ADDRS \triangleq LADDRS \cup HADDRS$

Zero data value (other than $ADDRS$ or $DATA$)
$Zero \triangleq \text{CHOOSE } val : val \notin ADDRS \cup DATA$

All values allowed in memory, registers
$VALUES \triangleq ADDRS \cup DATA \cup \{Zero\}$

Tables of operations ($e.g.$ arithmetic) on addresses and data. This is a reduced set so that model checking is still possible
$$optables \triangleq \{l \mathbin{@@} h : l \in [LADDRS \times DATA \rightarrow LADDRS],\ h \in [HADDRS \times DATA \rightarrow HADDRS]\}$$

TLC symmetry optimisations
$$Perms \triangleq Permutations(LADDRS) \cup Permutations(HADDRS) \cup Permutations(REGS) \cup Permutations(DATA)$$

MODULE *SimpleCPU*

Simple CPU implementation, no speculative execution

VARIABLES

$regs$,	CPU registers
$mode$,	security mode (low/high)
mem,	maps addresses to $VALUES$
$cached$	maps addresses to cached state

```
$vars \triangleq \langle regs, mode, mem, cached \rangle$

$TypeOK \triangleq$
        $\land\ regs \in [REGS \rightarrow VALUES]$
        $\land\ mode \in \{\text{“low”}, \text{“high”}\}$
        $\land\ mem \in [ADDRS \rightarrow VALUES]$
        $\land\ cached \in [ADDRS \rightarrow \text{BOOLEAN}]$
```

True if the memory at the given address is accessible in the current mode

```
$AccessOK(addr) \triangleq$
        $\text{IF } mode = \text{“low”} \text{ THEN } addr \in LADDRS \text{ ELSE } addr \in ADDRS$
```

CPU instructions

Set a register to a value

```
$SET(reg, val) \triangleq$
        $\land\ regs' = [regs \text{ EXCEPT } ![reg] = val]$
        $\land\ \text{UNCHANGED } \langle mode, mem, cached \rangle$
```

Copy the value in a register to another register

```
$MOV(regt, regm) \triangleq$
        $\land\ regs' = [regs \text{ EXCEPT } ![regt] = regs[regm]]$
        $\land\ \text{UNCHANGED } \langle mode, mem, cached \rangle$
```

Load a value from memory at the address in *regm* and store it in *regt*. Only enabled if the access is permitted in the current mode

```
$LDR(regt, regm) \triangleq$
        $\land\ \text{LET } addr \triangleq regs[regm]$
          $\text{IN}$      $\land\ AccessOK(addr)$
                   $\land\ regs' = [regs \text{ EXCEPT } ![regt] = mem[addr]]$
                   $\land\ cached' = [cached \text{ EXCEPT } ![addr] = \text{TRUE}]$
        $\land\ \text{UNCHANGED } \langle mode, mem \rangle$
```

Store the value in *regt* to memory at the address in *regm*. Only enabled if the access is permitted in the current mode

```
$STR(regt, regm) \triangleq$
        $\land\ \text{LET } addr \triangleq regs[regm]$
          $\text{IN}$      $\land\ AccessOK(addr)$
                   $\land\ mem' = [mem \text{ EXCEPT } ![addr] = regs[regt]]$
                   $\land\ cached' = [cached \text{ EXCEPT } ![addr] = \text{TRUE}]$
        $\land\ \text{UNCHANGED } \langle regs, mode \rangle$
```

Operation on register values with result in a destination register

```
$OP(regt, regm, regn, optable) \triangleq$
        $\land\ \text{LET } rm \triangleq regs[regm]$
                $rn \triangleq regs[regn]$
          $\text{IN}$      $\land\ \langle rm, rn \rangle \in \text{DOMAIN } optable$
```

```
            $\land\ regs' = [regs\ \text{EXCEPT}\ ![regt] = optable[\langle rm,\ rn\rangle]]$
        $\land\ \text{UNCHANGED}\ \langle mode,\ mem,\ cached\rangle$
```

Switching security/privilege modes (*e.g.* system call and return)

```
$HCALL \triangleq$
        $\land\ mode = \text{"low"}$
        $\land\ mode' = \text{"high"}$
        $\land\ \text{UNCHANGED}\ \langle regs,\ mem,\ cached\rangle$

$LRET \triangleq$
        $\land\ mode = \text{"high"}$
        $\land\ mode' = \text{"low"}$
        $\land\ \text{UNCHANGED}\ \langle regs,\ mem,\ cached\rangle$
```

Execute/dispatch an instruction in the form ⟨"*name*", *arg*1, *arg*2, ...⟩

```
$Exec(inst) \triangleq$
        CASE $inst[1] = \text{"set"}$    $\rightarrow SET(inst[2],\ inst[3])$
          □  $inst[1] = \text{"mov"}$    $\rightarrow MOV(inst[2],\ inst[3])$
          □  $inst[1] = \text{"ldr"}$    $\rightarrow LDR(inst[2],\ inst[3])$
          □  $inst[1] = \text{"str"}$    $\rightarrow STR(inst[2],\ inst[3])$
          □  $inst[1] = \text{"op"}$     $\rightarrow OP(inst[2],\ inst[3],\ inst[4],\ inst[5])$
          □  $inst[1] = \text{"hcall"}$  $\rightarrow HCALL$
          □  $inst[1] = \text{"lret"}$   $\rightarrow LRET$
          □  OTHER                       $\rightarrow Assert(\text{FALSE},\ \langle\text{"Unknown instruction"},\ inst[1]\rangle)$
```

MODULE *CPU*

CPU model together with its speculative state. The speculative state shares the memory and cache with the normal (committed) one, however, the speculative register bank is separate and not visible to the normally executing instructions.

```
VARIABLES
        $regs$,          CPU registers
        $specregs$,      speculative registers
        $mode$,          security mode
        $mem$,           maps addresses to VALUES
        $cached$         maps addresses to cached state

$vars \triangleq \langle regs,\ specregs,\ mode,\ mem,\ cached\rangle$

$ExecCPU \triangleq$ INSTANCE $SimpleCPU$
$SpecCPU \triangleq$ INSTANCE $SimpleCPU$ WITH $regs \leftarrow specregs$

$TypeOK \triangleq$
        $\land\ ExecCPU!TypeOK$
        $\land\ SpecCPU!TypeOK$
        $\land\ specregs \in [REGS \rightarrow VALUES]$

$Init \triangleq$
```

$\wedge\ mode = MODE$

if $mode$ = "high", $regs$ contain the low-provided input

$\wedge\ regs \in [REGS \rightarrow VALUES \setminus HADDRS]$

$\wedge\ specregs = regs$

Initial memory contains only $DATA$ or $Zero$ to reduce the number of initial states

$\wedge\ mem \in [ADDRS \rightarrow DATA \cup \{Zero\}]$

Cache empty initially

$\wedge\ cached = [a \in ADDRS \mapsto \text{FALSE}]$

Low/High states and low observation function. The low observation function exposes the cached state

$LowState \triangleq \langle regs, [addr \in LADDRS \mapsto mem[addr]]\rangle$

$HighState \triangleq \langle regs, [addr \in HADDRS \mapsto mem[addr]]\rangle$

$LowObs \triangleq [addr \in LADDRS \mapsto \langle mem[addr], cached[addr]\rangle]$

Normal execution discards the speculative registers

$Exec(inst) \triangleq ExecCPU!Exec(inst) \wedge specregs' = regs'$

Speculation leaves visible CPU registers and memory unchanged

$Spec(inst) \triangleq SpecCPU!Exec(inst) \wedge \text{UNCHANGED}\ \langle regs, mem\rangle$

Confidentiality property is modelled as an observation function identical for two system behaviours

VARIABLES

$regs1,\ specregs1,\ mem1,\ cached1,\ mode1,$	1st CPU state
$regs2,\ specregs2,\ mem2,\ cached2,\ mode2,$	2nd CPU state
cmd	last instruction (debug)

Logging for easier trace analysis

$LogCmd(c) \triangleq \text{IF}\ LOG\ \text{THEN}\ cmd' = c\ \text{ELSE}\ \ \text{UNCHANGED}\ cmd$

Set a register to any address (corresponding to the current mode) or data value. For high security addresses, in addition, ensure that the values at the corresponding address is identical in two execution traces. Note that we don't allow the full range of values while in "high" mode to be able to isolate the speculation side-channels triggered by the "low" mode state ("high" mode input). IOW, assume that the high security program has been hardened against non-speculative leaks.

$SafeHAddrs \triangleq \{a \in HADDRS : mem1[a] = mem2[a]\}$

```
HavocInst ≜
    {⟨"set", reg, val⟩ :
        reg ∈ REGS,
        val ∈ IF mode1 = "low"
                THEN LADDRS ∪ DATA
                ELSE SafeHAddrs}
```

Copy a register value to another register

$MoveInst \triangleq$

$\{\langle \text{"mov"}, regt, regm\rangle : regt, regm \in REGS\}$

Set a register to a value loaded from memory

$LoadInst \triangleq$

$\{\langle \text{"ldr"}, regt, regm\rangle : regt, regm \in REGS\}$

Store a register value to memory

$StoreInst \triangleq$

$\{\langle \text{"str"}, regt, regm\rangle : regt, regm \in REGS\}$

Compute (*regm op regn*) according to the given operation table and store the result in *regt*

$OpInst \triangleq$

$\{\langle \text{"op"}, regt, regm, regn, optable\rangle : regt, regm, regn \in REGS,$
$\quad optable \in optables\}$

Change of security level instructions. If the initial mode is "high", only model the return to the "low" mode

$ExcInst \triangleq$

```
IF $MODE = \text{"low"}$
 THEN $\{\langle \text{"hcall"}\rangle, \langle \text{"lret"}\rangle\}$
 ELSE $\{\langle \text{"lret"}\rangle\}$
```

The union of allowed *CPU* instructions under normal execution

$ExecInstructions \triangleq$

$HavocInst \cup MoveInst \cup ExcInst \cup LoadInst \cup StoreInst$

The union of instructions that can be speculatively executed. Not all set of instructions available to the speculating machine

$SpecInstructions \triangleq$

$LoadInst \cup OpInst$

Two *CPUs* used to model two separate execution traces

$CPU1 \triangleq \text{INSTANCE } CPU \text{ WITH } regs \leftarrow regs1, specregs \leftarrow specregs1, mode \leftarrow mode1,$
$\quad mem \leftarrow mem1, cached \leftarrow cached1$

$CPU2 \triangleq \text{INSTANCE } CPU \text{ WITH } regs \leftarrow regs2, specregs \leftarrow specregs2, mode \leftarrow mode2,$
$\quad mem \leftarrow mem2, cached \leftarrow cached2$

$vars \triangleq \langle CPU1!vars, CPU2!vars\rangle$

$TypeOK \triangleq CPU1!TypeOK \wedge CPU2!TypeOK$

$Init \triangleq$

$\wedge CPU1!Init \wedge CPU2!Init$
$\wedge CPU1!LowState = CPU2!LowState$
$\wedge CPU1!LowObs = CPU2!LowObs$

Reduce the initial state space for shorter *TLC* checking time

$\wedge \forall a \in LADDRS : mem1[a] = Zero$

$\land \forall a \in HADDRS : mem1[a] = Zero \lor mem1[a] \neq mem2[a]$

Debug log

$\land\ cmd = \langle\rangle$

Execute the same instruction deterministically on two *CPUs*. A deterministic algorithm (under no speculation) has the same register values in two separate execution traces

$ExecCPUNext \triangleq$

$\quad \exists\, inst \in ExecInstructions :$

$\quad\quad \land\ CPU1!Exec(inst)$

$\quad\quad \land\ CPU2!Exec(inst)$

$\quad\quad \land\ regs1' = regs2'$

$\quad\quad \land\ LogCmd(\langle \text{"exec:"} \rangle \circ inst)$

Speculatively execute the same instructions on two *CPUs*. Since the speculative registers are not part of the algorithmic state, they may differ in two separate execution traces

$SpecCPUNext \triangleq$

$\quad \exists\, inst \in SpecInstructions :$

$\quad\quad \land\ CPU1!Spec(inst)$

$\quad\quad \land\ CPU2!Spec(inst)$

$\quad\quad \land\ LogCmd(\langle \text{"spec:"} \rangle \circ inst)$

The next step consists of either a normally executed instruction or a speculative one

$Next \triangleq ExecCPUNext \lor SpecCPUNext$

$Spec \triangleq Init \land \Box[Next]_{\langle vars,\, cmd\rangle}$

The confidentiality property ensures that the low-observable state (low memory data and cached state) is the same in two separate execution traces. In other words, the low security observation is a deterministic function of only the initial register state (input to the high security mode), algorithmic state (deterministic in-memory values) and executed instructions. Any other non-determinstic high security state should not affect the observation function.

$ConfSideChannels \triangleq$

$\quad CPU1!LowObs = CPU2!LowObs$

THEOREM $Spec \Rightarrow \Box(TypeOK \land ConfSideChannels)$