Section 1

Microarchitecture Side-Channel Resistant Instructions Spans (scrispans)

Examples semantics

- `spansec.create [policies]`: create a new scrispan with given optional set of security policies (flags).
- `spansec.save rd`: save the current scrispan configuration (ID + security policy) in a register.
- `spansec.restore rs1`: restore a scrispan configuration from a register holding ID and security policy.

We assume that a change of scrispan ID implies microarchitectural isolation.

- `spansec.alter [policies]`: in addition we could add an instruction that keeps the ID unchanged but modify the security policies.

We are not discussing the implementation for now. Have you alternative semantics in mind ?

Security policies

`[policies]` is the set of security policies applied to a scrispan.

- `no speculation`
- `inline memory encryption`
- `constant time execution`

Personal comment (Ronan): security policies add complexity, but not sure if useful. Suggested policies could be better achieved with other means.

Do you have strong use cases for security policies ?

Usecase #1: web server

Goal: isolate users

Each process has its own scrispan.

```
// creating process P1
spansec.create

// during switch from P1 to P2
spansec.save P1
spansec.restore P2
```

Comments

- ✓ This explicit span gymnastic ensures proper microarchitectural state isolation.
- ✗ In this precise use case, we could tie the scrispan to the ASID (Address Space ID) instead.

Pattern: Microarchitectural state poisoning

Goal: isolating user poisoned uarch state from data processing

```
// span A: user controlled, span B: data processing
spansec.restore A
state_poisoning();
spansec.save A

spansec.restore B
data_processing();
spansec.save B
```

Comments

Since poisoning can be done in the same or another address space (cf `https://transient.fail`), we cannot rely on tying the scrispan with the ASID.

Simultaneous multithreading (SMT) I

Example from `https://github.com/IAIK/transientfail/blob/master/pocs/spectre/RSB/sa_ip/main.c`, two software threads in the same address space.

Pattern

```
pthread_t attacker_thread;
pthread_t victim_thread;
pthread_create(&attacker_thread, 0, attacker, 0);
pthread_create(&victim_thread, 0, victim, 0);
```

Solution

```
// pthread_create should start with
spansec.create
```

Simultaneous multithreading (SMT) II

Hardware consequences

- They are scheduled on different cores.
- If the hardware allows, the harts are isolated in the same core.
- They are not executed in parallel but sequentially, with proper microarchitectural flushing when switching between threads.

Spectre-PHT (v1) I

Vulnerabiliy

```
if (x < array1_size) {
    y = array2[array1[x] * 4096];
}
//speculative load 1: array1[x] -> accessing secret
//speculative load 2: array2[...] -> leaking gadget
```

Solution 1: disabling speculation

```
spansec.alter +nospeculation
if (x < array1_size) {
    y = array2[array1[x] * 4096];
}
spansec.alter -nospeculation
```

Spectre-PHT (v1) II

Solution 2: isolating poisoned state

```
if (x < array1_size) {
    spansec.save A
    spansec.create
    y = array2[array1[x] * 4096];
    spansec.restore A
}
```

Comments

- ✗ `nospeculation`: poor portability, precise behaviour is very hardware dependent, overkill ? (How to allow speculation across scrispan ?).
- ✗ `state isolation`: very bad performances.
- Probably needs another mechanism.