- title
- background
  - processors are machines to manipulate data, but to make this fast there are many optimisations that take place
    - not all physical processor elements are used at once. to increase usage, the processor works beforehand, and then check its work (speculative execution)
      - go through diagram, with concrete examples of what happens with the code.
      - mention *state* change without visible data change
    - branch predictor: a physical thing that remembers processor decisions and helps inform out-of-order
  - basic framework of speculative execution attacks
    - focusing on out-of-order
    - trigger speculation: attempt to do something that the processor recognises it might be able to do ahead of time. may be fine, but...
    - transient execution: on misspeculation, since the data is reverted but not the state, executed instructions can hide data in state
      - memory access privileges aren't checked during speculative execution
    - finally, go back and search the state for the data (recover secret)
  - canella's paper
    - classifies previous research about speculative execution attacks
    - interested in spectre, which uses the branch predictor to trigger misspeculation
- motivation
  - gem5 simulator
    - modular processor simulator with ability to model a wide variety of systems
    - enables finer control and monitoring than execution on a real processor, albeit without fully realistic behaviour
  - speculative execution attacks target frequently used elements in code and are untraceable (transients erase data)
    - programs may accidentally trigger misspeculation on their own and change processor state in an observable way, or an attacker's

code may look "normal"
    - goals: collate / create software to test for spectre vulnerabilities on both gem5 and a native environment
    - standard software for these vulnerabilities enables testing of mitigations (out of scope of this paper) and future architectures
- methods
    - searched for code to use as a basis online: canella had examples, other proof-of-concepts online
    - not all examples worked initially on gem5, because simulator is idealised and simplified
    - used the examples i had, knowledge about how a processor works, and the simulator's information to fix the code, make sure it's correct, and use it
    - canella's paper (revisited)
        - further breaking down speculation trigger into how to manipulate the history, and what physical element is being tricked
        - setup example: each process has its own virtual address space. branch_1 both creates the history and uses the misspeculation, branch predictor indexes based on virtual address.
        - canella's paper discusses ways of exploiting the function of the branch predictor to create a history that will result in a misspeculation.
        - looked at the gem5 code to inform selection of starting code to target following elements (explain code in greater detail)
            - pattern history table: whether to enter a branch
            - branch target buffer: where to enter after a branch: function pointer
            - return stack buffer: where to exit after a branch
- results / contributions
    - working, interchangeable code to test vulnerabilities on three parts of the branch predictor
    - works on both the simulator and an actual computer, and combines all three attacks into one program
    - implemented a RSB attack on gem5 after investigation of how the element in gem5 works
- conclusions / limitations
    - lower-level programming: more direct interactions with cpu and system

    - other training schemes, stl are not tested: code only tests one of the training schemes identified by canella due to limitations of the gem5 platform and time
        - gem5 does not share branch prediction elements between logical processors like in real CPUs, and does not have to speculate about address translation, so have to work around this
    - additional performance data that might be helpful to certain researchers, i.e. to determine if a mitigation reduces the bandwidth of or blocks an attack.
- acknowledgements