1. Motivation
- Improving current isolation mechanism performance
  - Guaranteeing same level of security
- Native Client
  - Software-based Fault Isolation implementation for CISC architectures
  - Instruction padding to enforce security policies
  - Padding imposes runtime overhead
  - We changed the padding scheme
  - Updated validator accordingly

2. Software-based Fault Isolation
- Applications may incorporate independently developed modules
  - Operating System
  - Add new file system
  - Database Management System: User-defined data type
- Problem with extensions
  - Security
  - Reliability
- Solution: Isolation
  - Load untrusted extension into its own fault domain
  - Code Segment
  - Data Segment
- Security Policy:
  - No code is executed outside of fault domain
  - No data is changed outside of fault domain

3. Google Native Client (NaCl)
- An SFI implementation for CISC and RISC architectures
- Allows execution of untrusted C/C++ code in Chrome browser
- Gives performance of native code to browser plugins
- On CISC architectures, it incorporates instruction padding to enforce an address layout invariant and restrict control flow
- Problem with variable size instructions in CISC architectures:
  - push $esi

4. Instruction Padding to Enforce Security Policies
- Unsafe instructions
  - jmp *%ecx
  - mov $0x123456, (%eax)
- Padding Scheme
  - Divide memory into 32-byte bundles (red boxes in the following listing)
  - Target of jumps placed at the beginning of bundles (type 1)
  - Call instructions placed at the end of bundles (type 2)
  - No instruction is allowed to cross bundle boundary (type 3)

5. Cross Bundle Instruction NaCl
- Change padding scheme
  - Allow instructions to cross the bundle boundaries as long as no unsafe instruction stream encountered
- Algorithms:
  - For each padding do the following:
    1. Set padding size to zero
    2. Assemble the binary
    3. If the validator fails on the binary, increase padding size by one
    4. Do the steps (2) and (3) until either validator succeeds or padding size reaches the original size
- Update the validator accordingly
  - We are allowing cross-bundle instructions in the binary
  - Validator must check no unsafe instructions are reachable
  - Multipass Validator: Start validation process from every cross point
  - Every bundle start
  - This way we can make sure every reachable address represents a valid instruction
  - We proved multipass validator correctness in Coq
  - Based on the RockSalt paper [G. Morrisett et al, PLDI 2012]

6. Evaluation
- We implemented our changes into GNU Assembler and NaCl validator
- Used SPECint CPU2000 as benchmark
- Number of instructions executed

7. Conclusion and Future Work
- We proposed more permissive padding policy
- Proved it is as secure as vanilla NaCl
- This optimization leads to decrease in the number of instructions executed and modest saving of averaging 1.5% in execution time
- Future work
  - Extending the CBI idea to x86-64 architecture
  - Replacing the greedy pad removal with a dynamic programming one