Exploiting and Protecting Dynamic Code Generation

Chengyu Song (GTISC),
Chao Zhang (UC Berkeley),
Tielei Wang, Wenke Lee (GTISC),
David Melski (Gramma Tech)

NDSS 2015

Agenda

- Motivation
  - W^X
  - Dynamic Code Generation (DCG)
- Exploiting DCG
  - A in-the-wild attack ➔ the threat is real and severe
  - A race-condition-based attack ➔ requires non-trivial protection
- Protecting DCG
  - A multi-process-based approach
    - **Secure, easy to adopt, low performance overhead**

Background

- W^X
  - Memory cannot be both Writable and eXecutable
  - **Effective** against code injection attack
  - **Efficient** with hardware support
- Dynamic Code Generation
  - Just-in-time (JIT) compilers
  - Dynamic binary translators
    - To enable dynamic analysis (e.g., PIN)
    - To provide portability (e.g., QEMU)
    - To help bug diagnosis (e.g., Valgrind)
    - To enhance security (e.g., ISR, ILR, DIFT)

The Problem

- Dynamic code generators usually keep code pages writable
  - For the ease of emitting new code
  - To patch existing code
    - For example, when a new code fragment is generated, existing code that will branch to this new fragment should be patched to improve the performance
- Unfortunately, this violates of the W^X principle and opens doors for attacks

An In-the-wild Exploit

- Mobile Pwn2Own Autumn 2013 – Chrome browser on Android
  1) Exploit an integer overflow vulnerability ➔ arbitrary read and write;
  2) Traverse memory and locate the code pages ➔ bypass ASLR;
  3) Leverage the arbitrary memory write capability to overwrite an JavaScript function with shellcode ➔ bypass guard pages;
  4) Invoke the JS function to execute the shellcode ➔ bypass CFI.

Security Implications

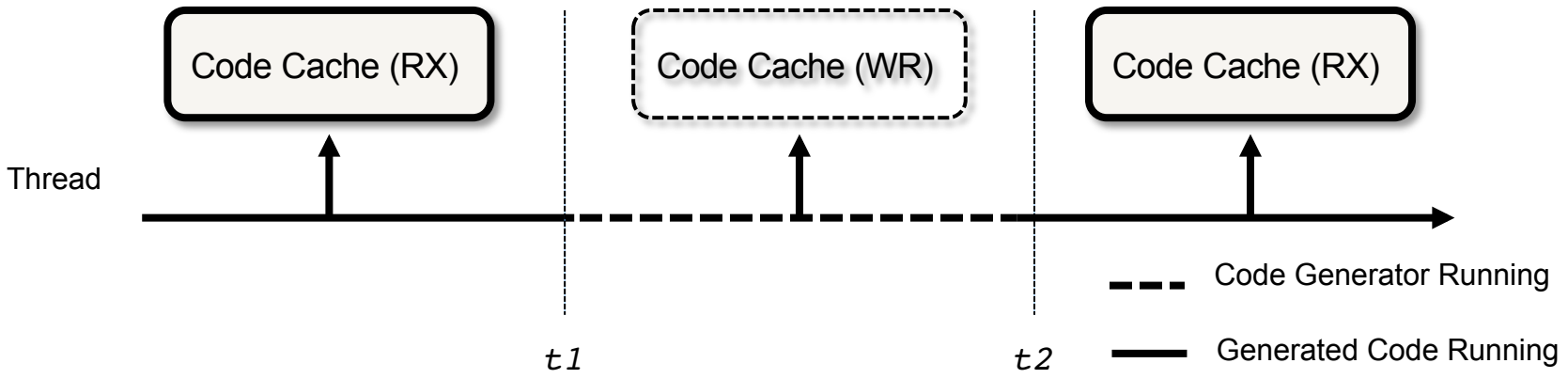
- Revives code injection attacks
- Breaks many defense mechanisms
  - CFI and ROP detection
    - Breaks the assumption that code will not deviate from the known control-flow graph
  - Any dynamic instrumentation based security solutions (e.g., dynamic taint analysis)
    - Injected code is not monitored
    - Existing checks can be removed

Feasibility of Such Attacks

- Bypassing ASLR
  - Brute force & spray attacks (32-bit platform);
  - Information disclosure vulnerability is widely available[1,2]
- Arbitrary memory write
  - Can be acquired from many types of vulnerabilities: integer overflow, format string, heap overflow, type confusion, use-after-free, etc.
  - Arbitrary memory read and write usually come together

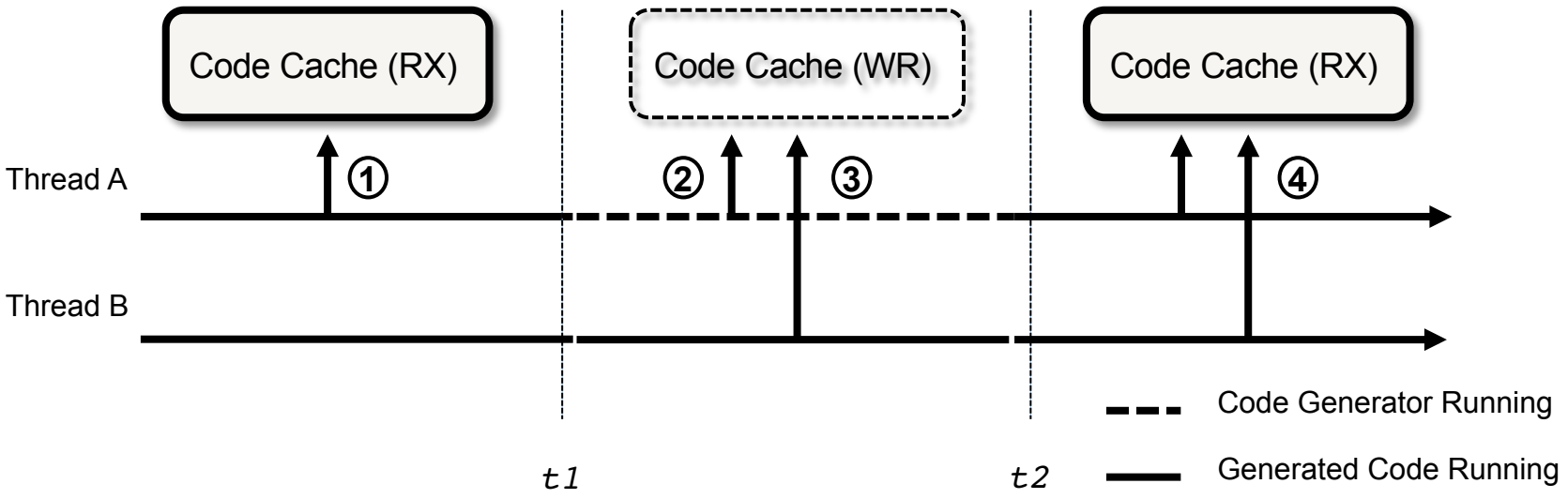
1. G. F. Roglia, L. Martignoni, R. Paleari, and D. Bruschi, Surgically returning to randomized libc
2. F. J. Serna, The info leak era on software exploitation

A Naïve Protection Idea



- Enforce that code pages can never be both writable and executable **at the same time**
  - Has been adopted by some JIT compilers
    - Mobile Safari
    - Internet Explorer

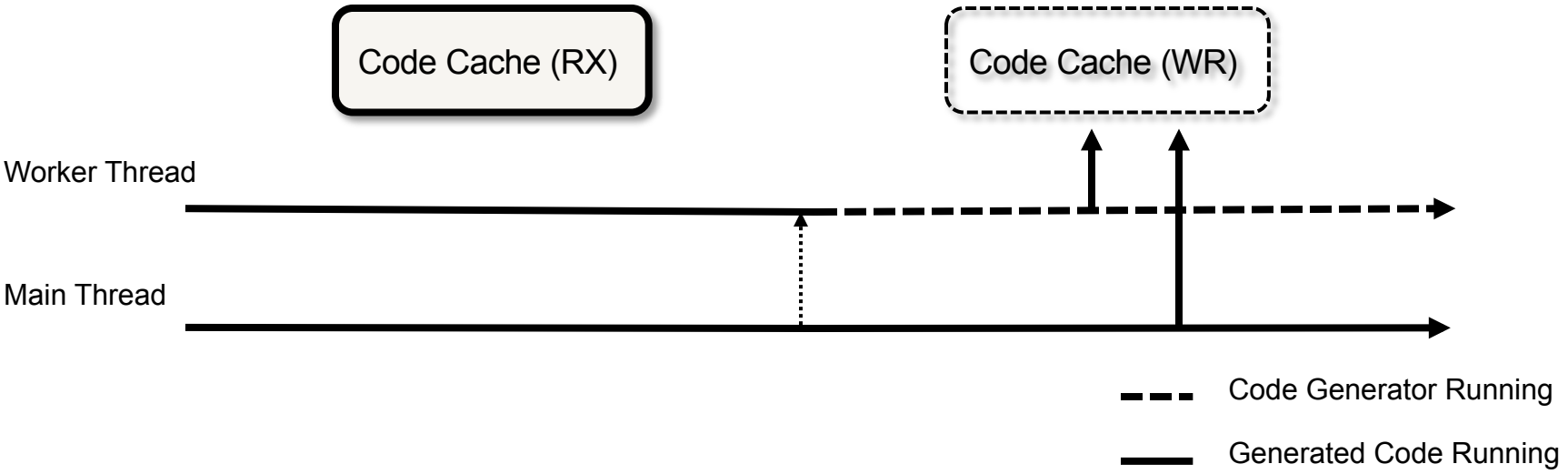
Exploiting Race Conditions



- Thread A cannot overwrite the code cache when the untrusted code is being executed (access 1);
- But when the code generator needs to modify the code cache (access 2);
- The code cache can then be overwritten by Thread B (access 3).
- Attack window: t1 ~ t2, access 4 will fail

A Proof-of-Concept Attack

- Exploiting V8 JS engine in the Chrome browser
  - Multi-thread programming through the `WebWorker` specification

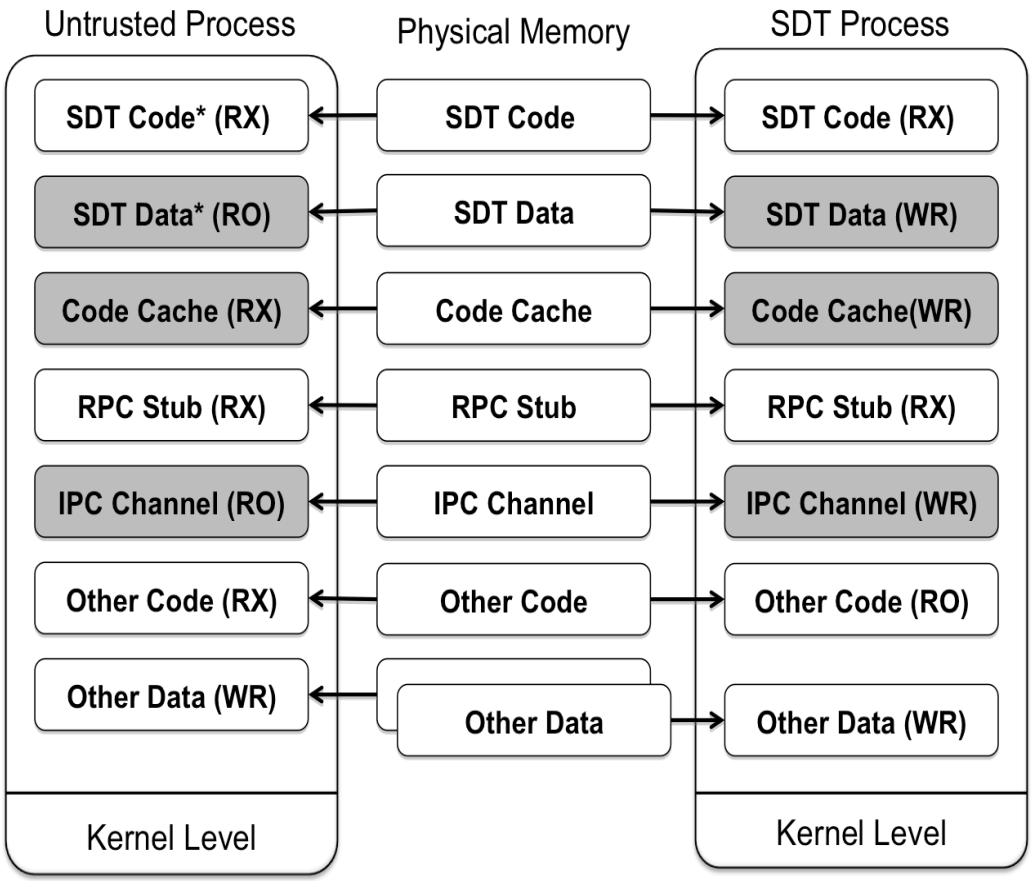


Reliability of Such Attacks

- Our PoC attack had a **91% success rate** (91/100)
- Thread synchronization latencies are usually smaller than the attack window
- Page access permission change can enlarge the attack window
  - The `mprotect` system call on Linux usually triggers the current thread be de-scheduled

Secure Dynamic Code Generation

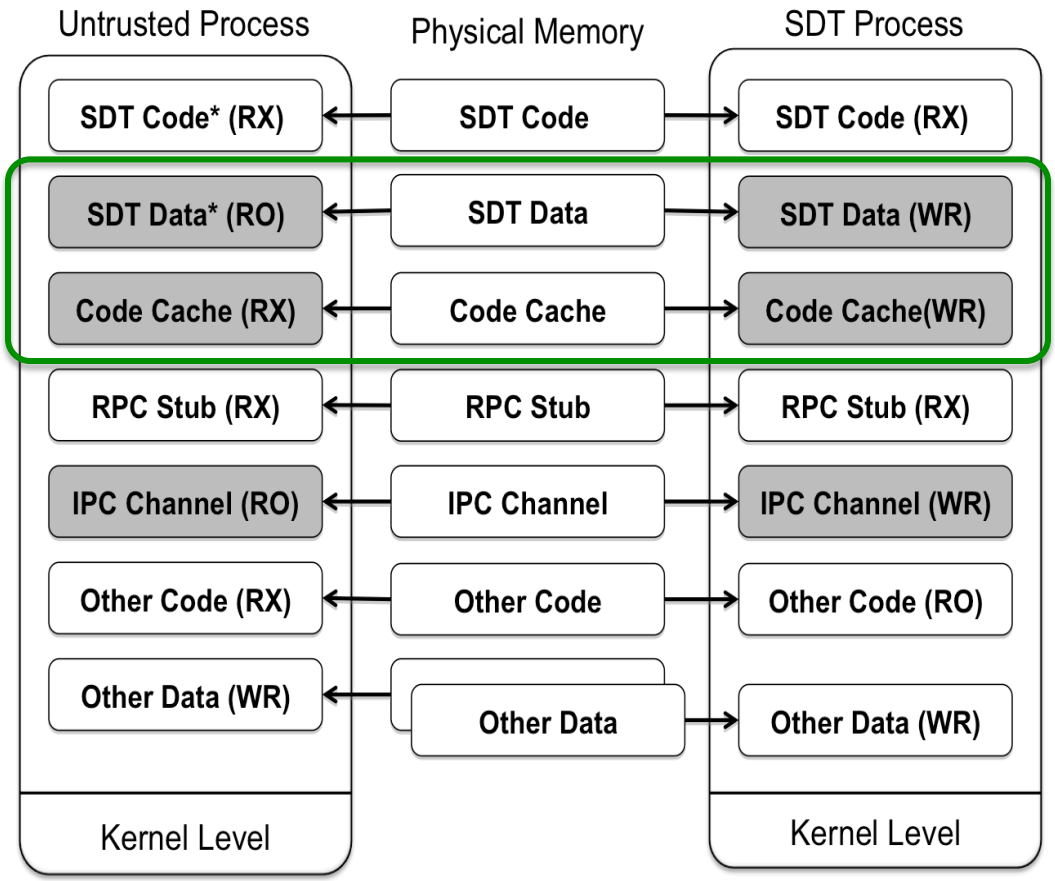
- A multi-process-based protection scheme
- Ensures code pages are permanently mapped as RX



SDT = software dynamic translator

Challenges (1)

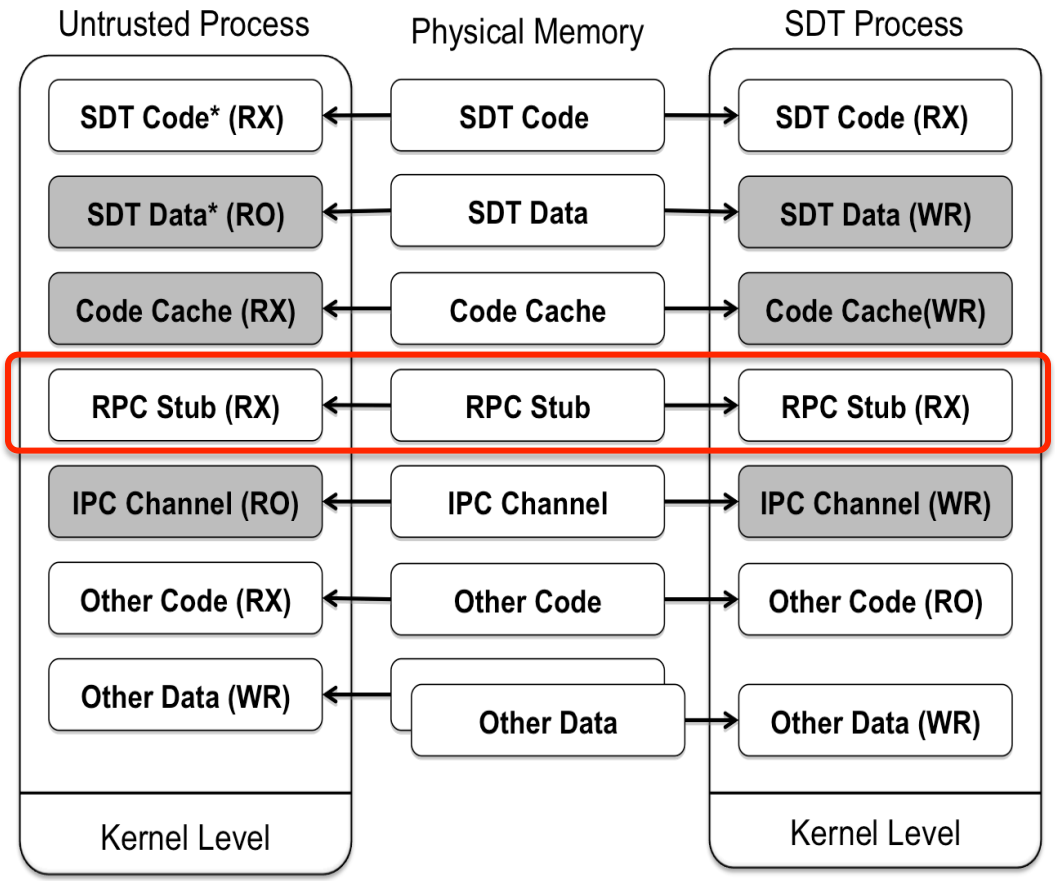
- Memory Map Synchronization
  - Shared resources have to be mapped at exactly the same memory address
- Solution
  - Shared memory pool



SDT = software dynamic translator

Challenges (2)

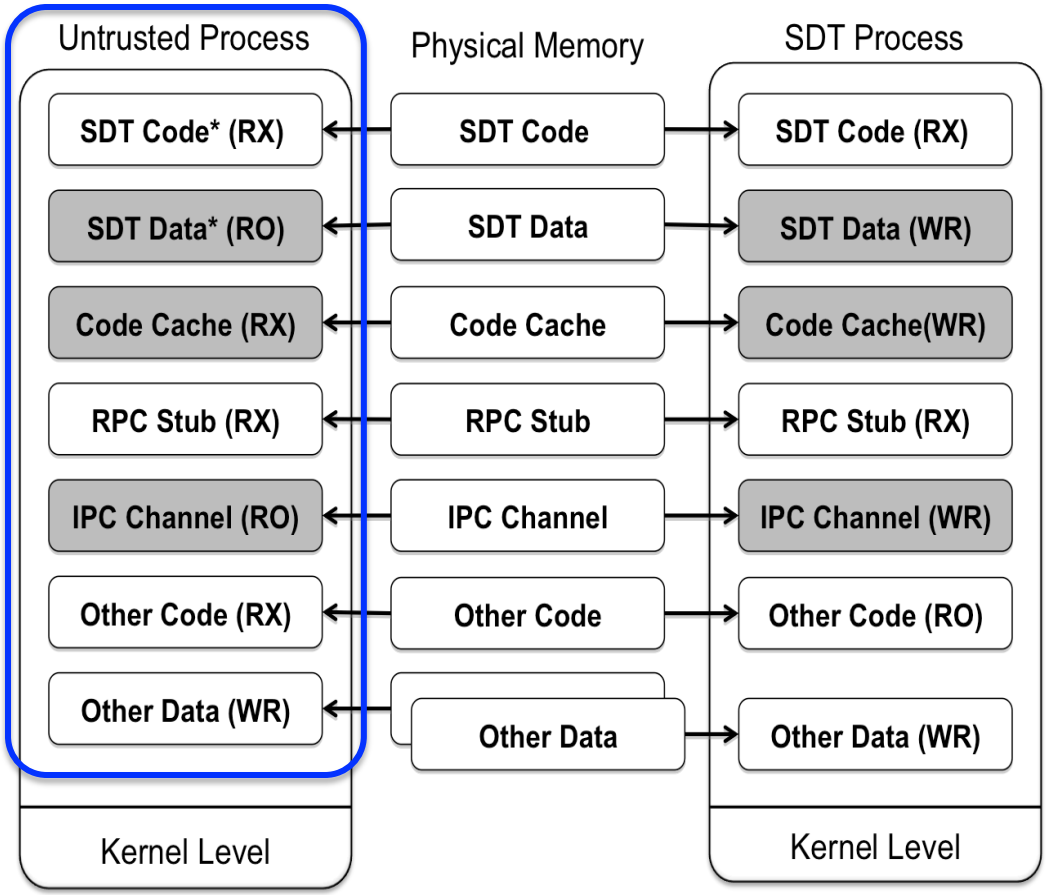
- Remote Procedure Call (RPC)
  - Argument passing
  - Invocation frequency
- Solution
  - Heap sharing + stack copy
  - Lazy stub generation



SDT = software dynamic translator

Challenges (3)

- Access Permission Enforcement
  - Untrusted code may try to tamper with the protection scheme
- Solution
  - System call interposition



SDT = software dynamic translator

Prototype Implementations

- Two Prototype implementations
  - Strata DBT and V8 JS engine on Linux
  - Sharable infrastructure (~500 LoC)
    - Shared memory pool
    - Trusted thread (based on seccomp-sandbox)
    - System call filtering
  - SDT-specific modification
    - Strata (~1000 LoC)
    - V8 (~2500 LoC)

Cache Coherency Overhead

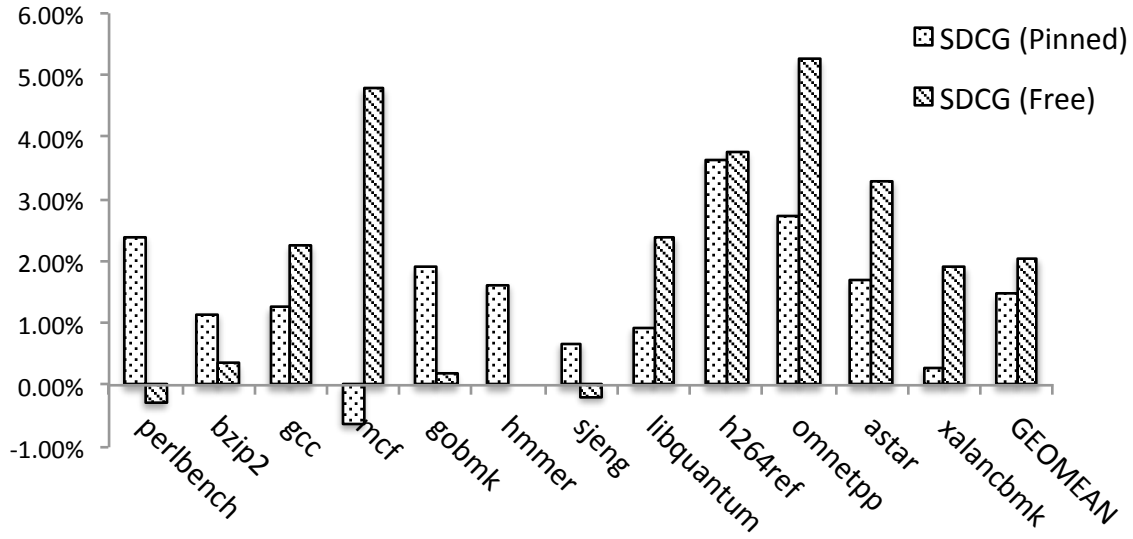
- 3 threads: untrusted main, SDT main, trusted
- 6 schedules: all pinned <-> all free
- Observation: schedule of **the two main threads has to be pinned together**, otherwise 3x-4x slower RPC invocation

TABLE II: Cache Coherency Overhead Under Different Scheduling Strategies.

	Schedule 1	Schedule 2	Schedule 3	Schedule 4	Schedule 5	Schedule 6
Richards	4.70 μs	13.76 μs	4.47 μs	14.25 μs	12.85 μs	13.37 μs
DeltaBlue	4.28 μs	13.29 μs	4.31 μs	13.85 μs	14.09 μs	15.84 μs
Crypto	3.99 μs	10.91 μs	3.98 μs	14.07 μs	12.47 μs	13.48 μs
RayTrace	3.98 μs	14.99 μs	4.05 μs	14.76 μs	13.15 μs	12.35 μs
EarlyBoyer	3.87 μs	13.70 μs	3.87 μs	14.27 μs	13.42 μs	13.47 μs
RegExp	3.82 μs	14.64 μs	3.85 μs	14.48 μs	13.55 μs	12.32 μs
Splay	4.63 μs	12.92 μs	4.49 μs	13.22 μs	13.36 μs	15.11 μs
NavierStokes	4.67 μs	12.06 μs	4.47 μs	13.02 μs	14.80 μs	12.65 μs

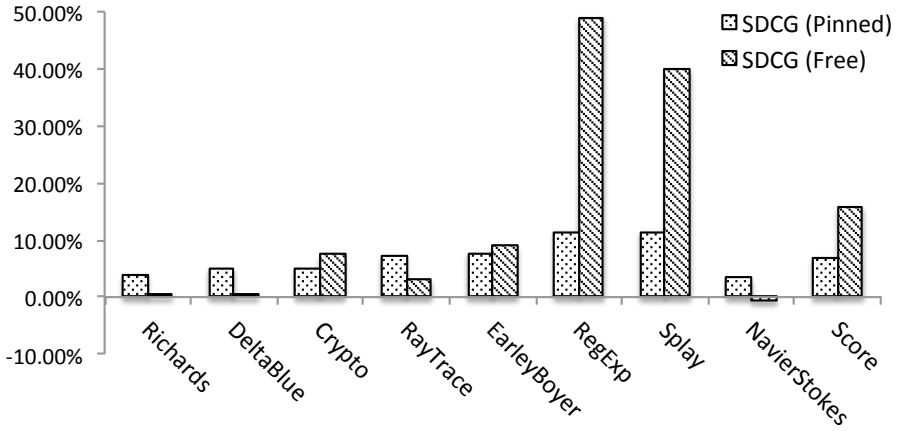
SPEC CINT 2006

- 1.46% for pinned schedule
- 2.05% for free schedule

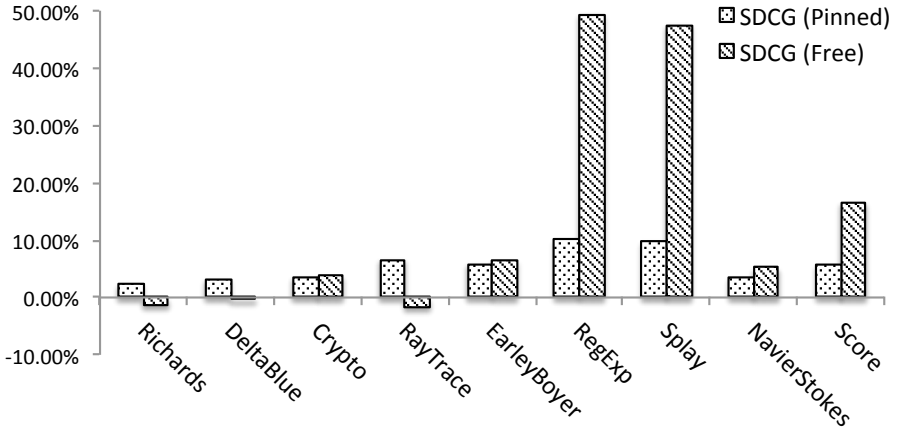


JavaScript Benchmark

- 6.9% for 32-bit build
- 5.65% for 64-bit build
- Comparison: NaCl-JIT (SFI) 79% for 32-bit build



32-bit build



64-bit build

Georgia Institute of Technology

Conclusion

- Dynamic code generation
  - Can be used as an attack vector to revive code injection attacks
  - Securing it is not trivial for multi-thread programs
  - We proposed Secure Dynamic Code Generation
    - Enforces mandatory W^X
    - Easy to adopt by existing software dynamic translators
    - Imposes small performance overhead

Thank you!