Paper Reading Group

Dec 13th, 2006

Evaluating SFI for a CISC Architecture

by

Stephen McCamant (MIT),
Greg Morrisett(Harvard University)

Reference

- This presentation is derived work. The original paper can be found at [2] as of Dec 2006. All pictures are directly copied from it. The project has also a home page [1].

  [1] http://pag.csail.mit.edu/~smcc/projects/pittsfield
  [2] http://pag.csail.mit.edu/~smcc/projects/pittsfield/pubs/usenix-sec-2006/pittsfield-usenix2006.pdf

Motivation

- untrusted code in secure systems
- SFI constrains actions to security policy by rewriting
- original SFI worked only on RISC
  - registers dedicated to SFI
  - instruction boundaries are well known

Other Options

- hardware memory protection
  - robust and low (runtime) overhead
  - coarse-grain and relatively expensive interaction across a process boundary
- type-safe languages
  - control flow and memory access limited to well-behaved patters by languages type discipline
  - limited to one language, no support for C/C++
  - no mechanisms to constrain code at machine instruction level
- SFI lies in between
  - security policy similar to OS
  - ahead-of-time verification of type system

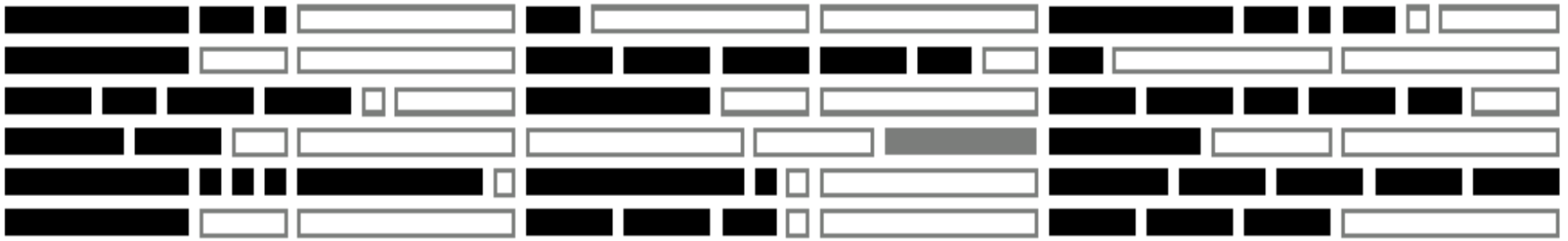
Classic SFI

- prevent potentially unsafe insn from being executed with improper arguments
- check pointers for validity
  - original SFI proposed naturally alligned regions (AND operation)
- ensure that checks cannot be bypassed
  - insert check before potentially unsafe operation
  - jump might redirect control flow behind the check
  - indirect jumps are the hard problem
    - switch, function pointers, object dispatch tables
  - Wahbe et all directed all operations through a dedicated register so that a jump to ANY instruction in the code region would be safe

CISC SFI

- Wahbe's approach not applicable for CISC
  - shortage of registers makes dedication undesirable (though only 2 instead of 5 would be needed)
- checks must precede critical instructions
- variable length instructions
  - it is infeasible to check ALL possible instruction streams
  - artificially enforce alignment
    - insertion of nops
    - control transfer is only allowed to the start of a chunk
      - call goes to the end of chunk so that the return enters at the beginning of the next one

Chunks



Optimizations

- special registers
  - handle ebp which is already constrained by the ABI as reserved register, i.e. check when reloaded
- guard regions
- ensure, do not check
- one-instruction address operation
  - address region tag contains only one bit set
  - zero tag region is unmapped
- efficient return
  - exploit call/return prediction in modern processors

Rewriting Example

```
f:  push    %ebp
    mov     %esp, %ebp
    mov     8(%ebp), %edx
    mov     48(%edx), %eax
    lea     1(%eax), %ecx




    mov     %ecx, 48(%edx)
    pop     %ebp



    ret
```

```
f:  push    %ebp
    mov     %esp, %ebp
    mov     8(%ebp), %edx
    mov     48(%edx), %eax
    lea     1(%eax), %ecx
    lea     0(%esi), %esi
    ---------------------
    lea     48(%edx), %ebx
    lea     0(%esi), %esi
    lea     0(%edi), %edi
    ---------------------
    and     $0x20ffffff, %ebx
    mov     %ecx, (%ebx)
    pop     %ebp
    lea     0(%esi), %esi
    ---------------------
    and     $0x20ffffff, %ebp
    andl    $0x10fffff0, (%esp)
    ret
```

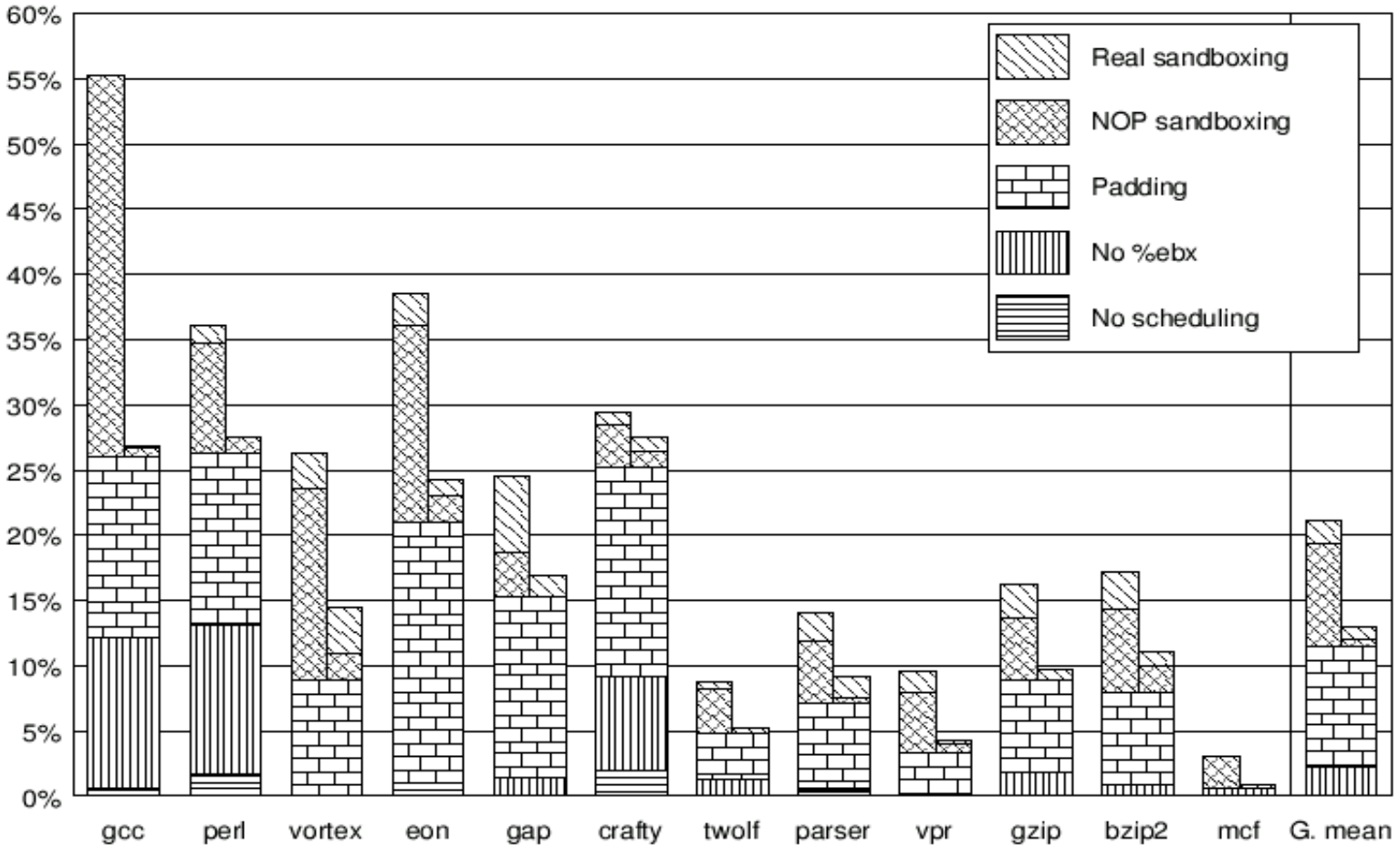
Verification

- compilation and rewriting are performed by untrusted producer
- safety policy is enforced by a separate verification tool
  - should be dependable (i.e. small)
  - check security properties
    - jumps never outside its code region
    - writes never outside its data region
  - not generally decidable
    - be conservative: allow false negatives but no false positives

Prototype

- rewriting tool
  - text processing tool
    - 720 LOC
    - operates on input to gas
  - one reserved register to hold the sandboxed address for both data and code regions
  - 64k guard regions
- verifier (2 versions)
  - text processing (700 LOC)
  - integrated in the object loader
    - check 2.7MB rewritten gcc in half a second

Performance Evaluation



Size Inflation

Program	gcc	perl	vortex	eon	gap	crafty	twolf	parser	vpr	gzip	bzip2	mcf
Size	2.7M	1.2M	1010K	923K	853K	408K	390K	276K	267K	109K	108K	50K
Ratio	1.84	1.96	1.63	1.72	1.84	1.62	1.80	1.92	1.67	1.65	1.63	1.74
Compressed	1.05	1.07	0.98	1.05	1.05	1.06	1.08	1.06	1.07	1.10	1.09	1.13

Application Case Study: VXA

	Zlib	BZip2	JPEG	JPEG2000	FLAC	Vorbis	Geom. Mean
VX32	1.006	0.975	1.034	1.283	0.954	0.948	1.028
PittSFIeld jump-only	1.238	1.018	1.134	1.114	1.142	1.239	1.145
PittSFIeld full	1.398	1.072	1.328	1.211	1.241	1.458	1.278

Formal Analysis

- ACL2 proof for verificator
- “if the verifier approves the rewritten code, then for any inputs, execution of the code will contineu forever without performing an unsafe operation”