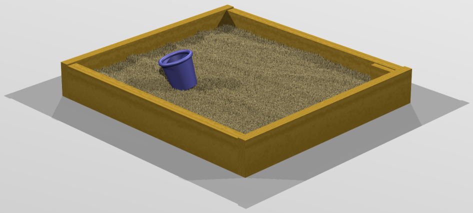


Evaluating SFI for a CISC Architecture

Stephen McCamant and Greg Morrisett
smcc@csail.mit.edu, greg@eecs.harvard.edu
MIT CSAIL and Harvard EECS

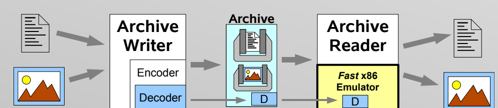
Outline

Software security: isolation

- How can I keep a piece of code from doing bad things?
- Author might be malicious, or code might be subverted by malicious input
- Identify legal interfaces; how to limit interaction to them?

Application: future-proof archives

- Embed decompressor in `.zip` file so it's always available [Ford, 2005]
- How to safely execute untrusted library?



Well-known isolation techniques

- OS process abstraction
  - \+ Robust hardware enforcement
  - – System-call interface inflexible
- Type-safe programming language (e.g., Java)
  - \+ Allows fine-grained data sharing
  - – Not applicable to C/C++

SFI in outline

- "Software-based Fault Isolation"
- Simulate hardware-style protection with binary-level rewriting
- Insert checks to confine jumps and memory writes to sandbox regions

Outline

- SFI as a security technique
- Classic (RISC) SFI
- A CISC-compatible approach
- PittSFIeld implementation
- Machine-checked proof
- Conclusion

Key problem: circumventing checks

```
f00:  check %rs
f04:  unsafe op %rs
 ⋮      ⋮
f80:  jmp f04
 ⋮      ⋮
fbc:  check-bounds %rt
fc0:  jmp %rt
```

- Do checks always precede unsafe ops?

Solution: dedicated registers

- Indirect write only through `%rs`
- Maintain invariant: at jump, `%rs` contains a legal data address
- Safe to jump into middle of checks

```
f40:  mov %rt -> %rs
f44:  check %rs
f48:  store %x, (%rs)
```

- Requires several registers

Bitwise memory isolation

- Distinct code and data areas to prevent self-modifying code
- Areas have power-of-two size and alignment
- Enforce by bitwise AND and OR on addresses

Ensure, don't check

- Ideal: if the original program would have violated the security policy, the transformed program will halt with an error message right before the violation.

Ensure, don't check

- Relaxed: if the original program would have violated the security policy, the transformed program will do something allowed by the security policy.

More optimizations

- Trusted register: check after modification, not before use
  - Invariant: frame pointer always safe for data region
- Guard pages: put unmapped pages at edges of data area
  - E.g., `push` needs no checks

Outline

SFI as a security technique

Classic (RISC) SFI

A CISC-compatible approach

PittSFIeld implementation

Machine-checked proof

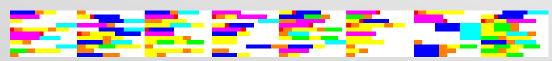
Conclusion

Key problem: overlapping instructions

```
          push %esi
     mov $0x56,%dh|sbb $0xff,%al|inc %eax|or %al,%dh
movzbl 0x1c(%esi),%edx|     incl 0x8(%eax)      ...
0f b6       56        1c     ff        40     08    c6
```

- Processor can jump to any byte
- Hard to make hidden instructions safe

Solution: enforce instruction alignment



- No instruction crosses a 16-byte boundary
- Jump targets have low 4 bits zero
- `call` instructions end on 16-byte boundaries
- Only need one spare register

Optimization: AND-only sandboxing

- Reduce sandboxing sequence to one instruction
- Mask address with `20ffffff`
- Reserve `00000000` to `00ffffff`

Security model

- Compiler and rewriter are untrusted
- Check rewriting on load; only this checker needs to be trusted
- Disallow unknown instructions
- Safety does not depend on compiler sanity

Outline

- SFI as a security technique
- Classic (RISC) SFI
- A CISC-compatible approach
- PittSFIeld implementation
- Machine-checked proof
- Conclusion

PittSFIeld

- Prototype IA-32 Transformation Tool for Software-based Fault Isolation Enabling Load-time Determinations (of safety)
- `http://pag.csail.mit.edu/~smcc/projects/pittsfield`
- Google: PittSFIeld SFI

Assembly-language rewriting

- Rewriter is a Perl program that operates on GAS assembly code
- Alignment using `.align` directives and conservative length estimation
- Important to rewrite before symbolic references resolved (done by code producer)

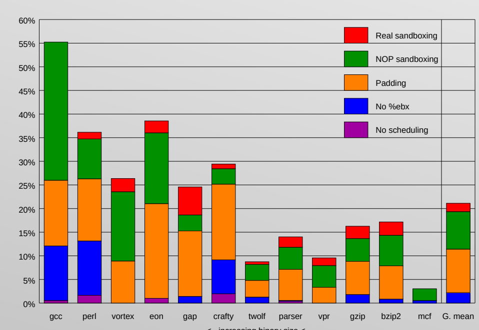
One-pass, local verification

- Single in-order pass over instruction sequence
- State machine keeps track of static invariant validity
  - Conservative assumptions at potential jump targets
  - Must clean up before jumping elsewhere

SPEC benchmarks (gcc = 1.0)

benchmark	time	size	compr. size
Geom. Mean	1.21	1.75	1.07
164.gzip	1.16	1.65	1.10
175.vpr	1.07	1.67	1.07
176.gcc	1.55	1.84	1.05
181.mcf	1.01	1.74	1.13
186.crafty	1.29	1.62	1.06
197.parser	1.14	1.92	1.06
252.eon	1.35	1.72	1.05
253.perlbmk	1.36	1.96	1.07
254.gap	1.24	1.84	1.05
255.vortex	1.23	1.63	0.98
256.bzip2	1.16	1.63	1.09
300.twolf	1.08	1.80	1.08

Sources of time overhead



Outline

SFI as a security technique

Classic (RISC) SFI

A CISC-compatible approach

PittSFIeld implementation

Machine-checked proof

Conclusion

One good basket

- For security, key is verifier
- Want to know that if verifier says OK, code is really safe
- Prove it!
- Machine-checked proof for increased assurance

ACL2

- ACL2 is a proof-assistant environment from J Moore et al. (UT Austin)
- Model a problem in restricted subset of Common Lisp
  - (no mutation, higher-order functions)
- Refine goal into small sub-lemmas, each proved automatically
  - (perhaps with 'hints')

Statement to prove

- Verifier implements a predicate on the code image
- Model the processor as an interpreter
- Unsafe operations cause it to halt, no `exit`
- $\forall$ code: (code passes verifier) $\Rightarrow$ (code runs forever)

Proof status

- Verified for a small but representative instruction subset:

```
nop          mov addr, %eax      xchg %eax, %ebx
inc %eax     mov %eax, addr      xchg %eax, %ebp
jmp addr     and $immed, %ebx    mov %eax, (%ebx)
jmp *%ebx    and $immed, %ebp    mov %eax, (%ebp)
```

- Realistic padding and encoding

Outline

SFI as a security technique

Classic (RISC) SFI

A CISC-compatible approach

PittSFIeld implementation

Machine-checked proof

Conclusion

Conclusion

- It is possible to do SFI efficiently on a CISC architecture
- It is possible to apply SFI to full-scale applications
- It is possible to trust an SFI implementation

Questions?