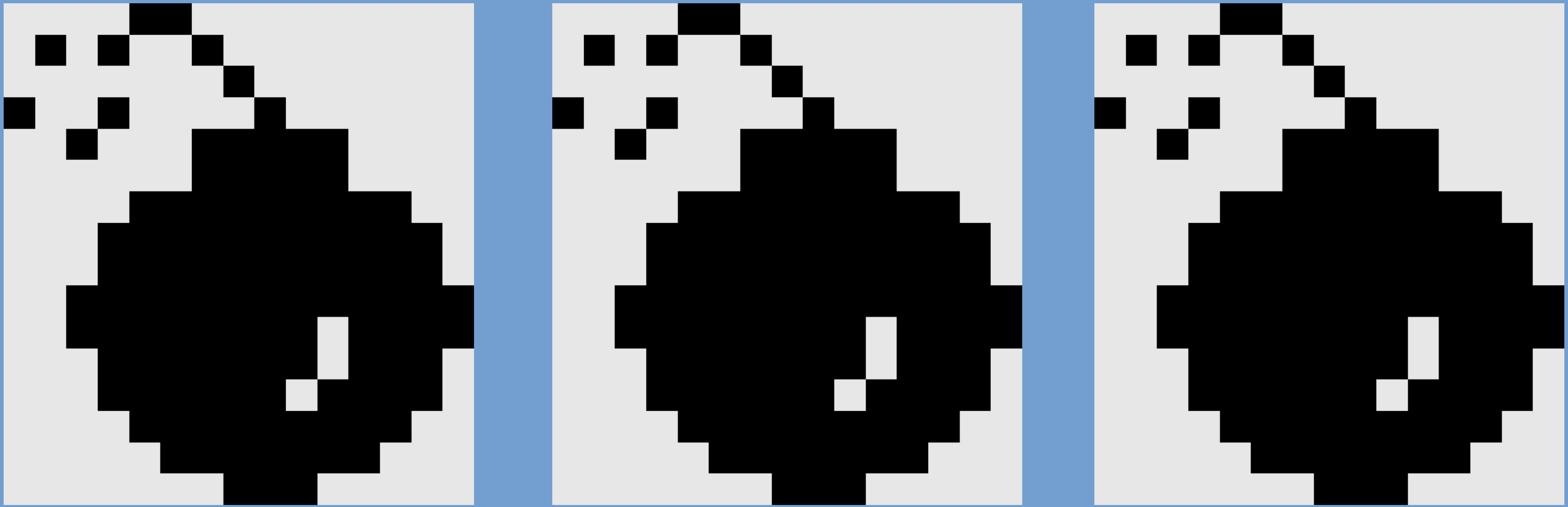


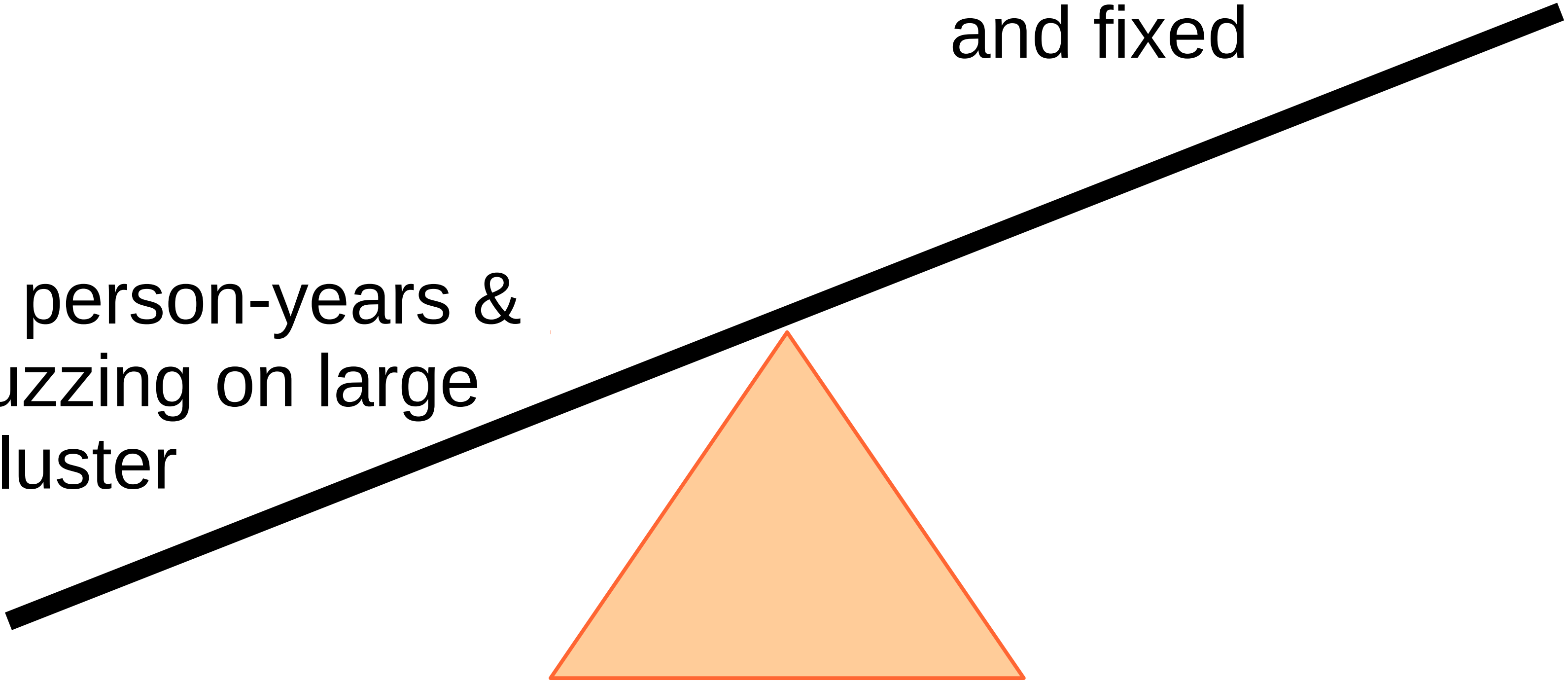
Fine-Grained Control-Flow Integrity through Binary Hardening

Mathias Payer, Antonio Barresi, Thomas R. Gross

PURDUE UNIVERSITY®

ETH Zürich

FFmpeg and a thousand fixes



>1,000 bugs found and fixed

2 person-years & fuzzing on large cluster

http://j00ru.vexillium.org/?p=2211 Jan-10, 2014

Software is unsafe and insecure

- Low-level languages (C/C++) trade type safety and memory safety for performance
  - Programmer responsible for all checks

- Large set of legacy and new applications written in C / C++ prone to memory bugs

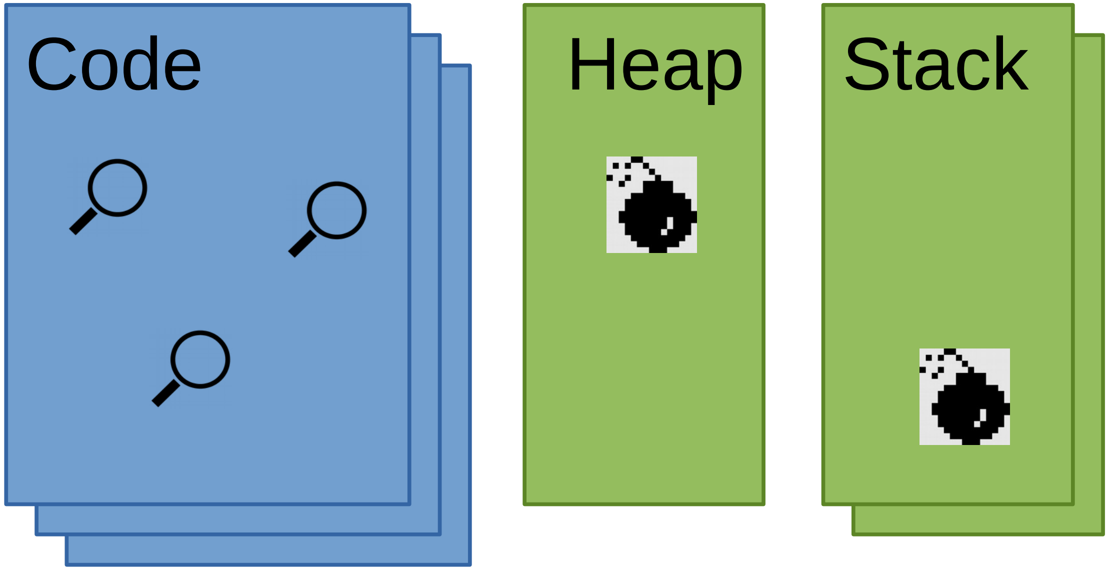
- Too many bugs to find and fix manually
  - Protect integrity through safe runtime system

Vulnerability
Just Ahead

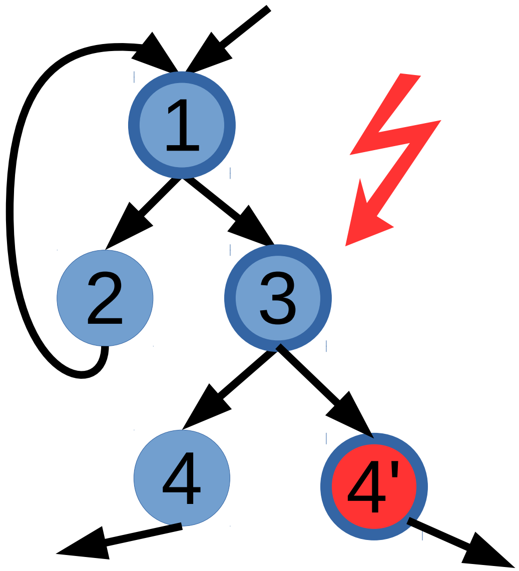
Code Reuse Attacks

Attack scenario: code reuse

- Find addresses of gadgets
- Force memory corruption to set up attack
- Leverage gadgets for code-reuse attack



Control-flow hijack attack



- Attacker modifies ***code pointer***
  - Function return
  - Indirect jump
  - Indirect call
- Control-flow leaves ***valid graph***
- Reuse existing code
  - Return-oriented programming
  - Jump-oriented programming

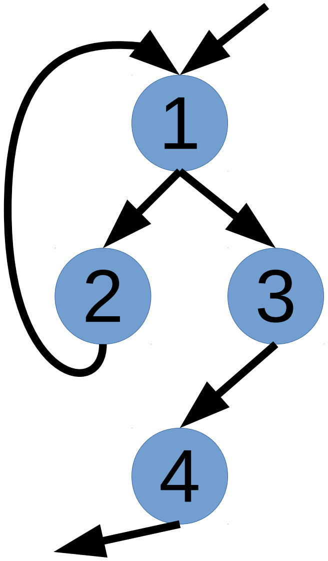
Control-Flow Integrity

Control-Flow Integrity (CFI)

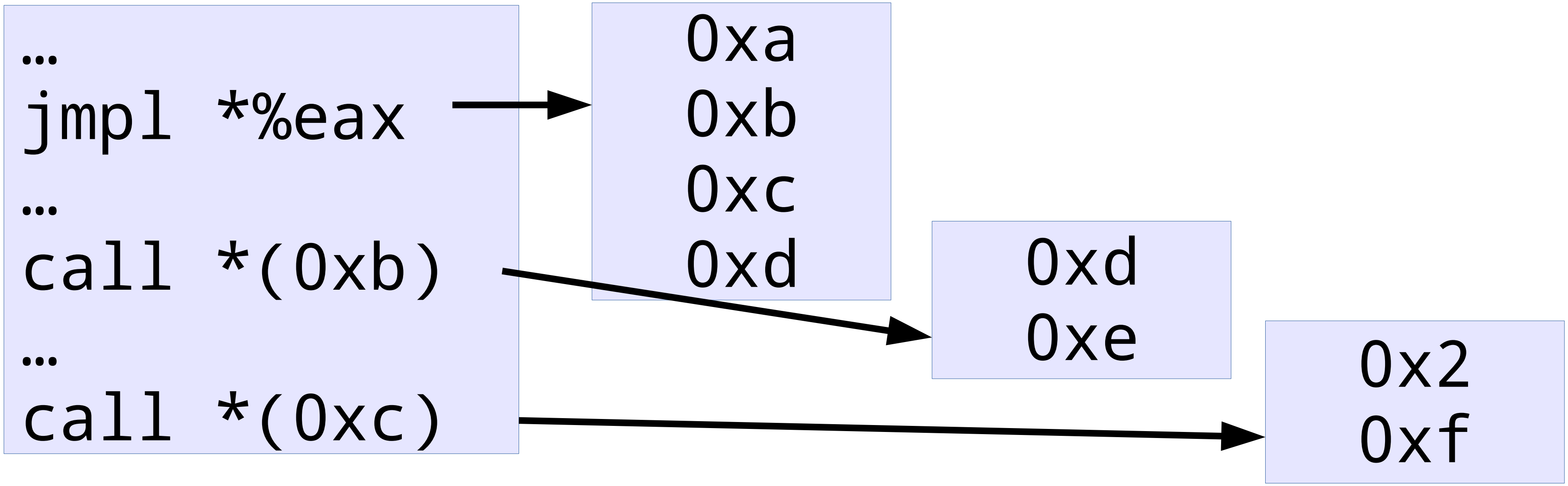
- CFI enforces that each dynamic indirect control flow transfer must target a statically determined set of locations
- Three sources of indirect transfers
  - Indirect jump
  - Indirect call
  - Function returns

Control-Flow Integrity (CFI)

- Statically construct Control-Flow Graph
  - Find set of allowed targets for each location



- Online set check



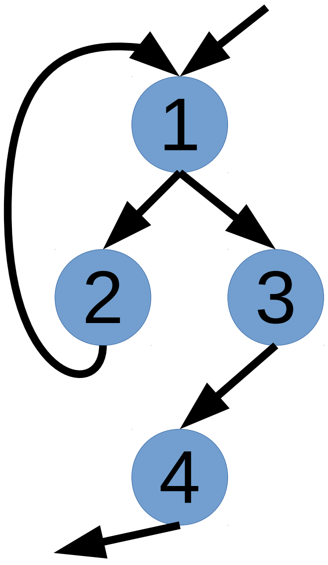
```
…
jmpl *%eax
…
call *(0xb)
…
call *(0xc)
```

```
0xa
0xb
0xc
0xd
```

```
0xd
0xe
```

```
0x2
0xf
```

Control-Flow Integrity (CFI)



- Statically construct Control-Flow Graph
  - Find set of allowed targets for each location

- Online set check

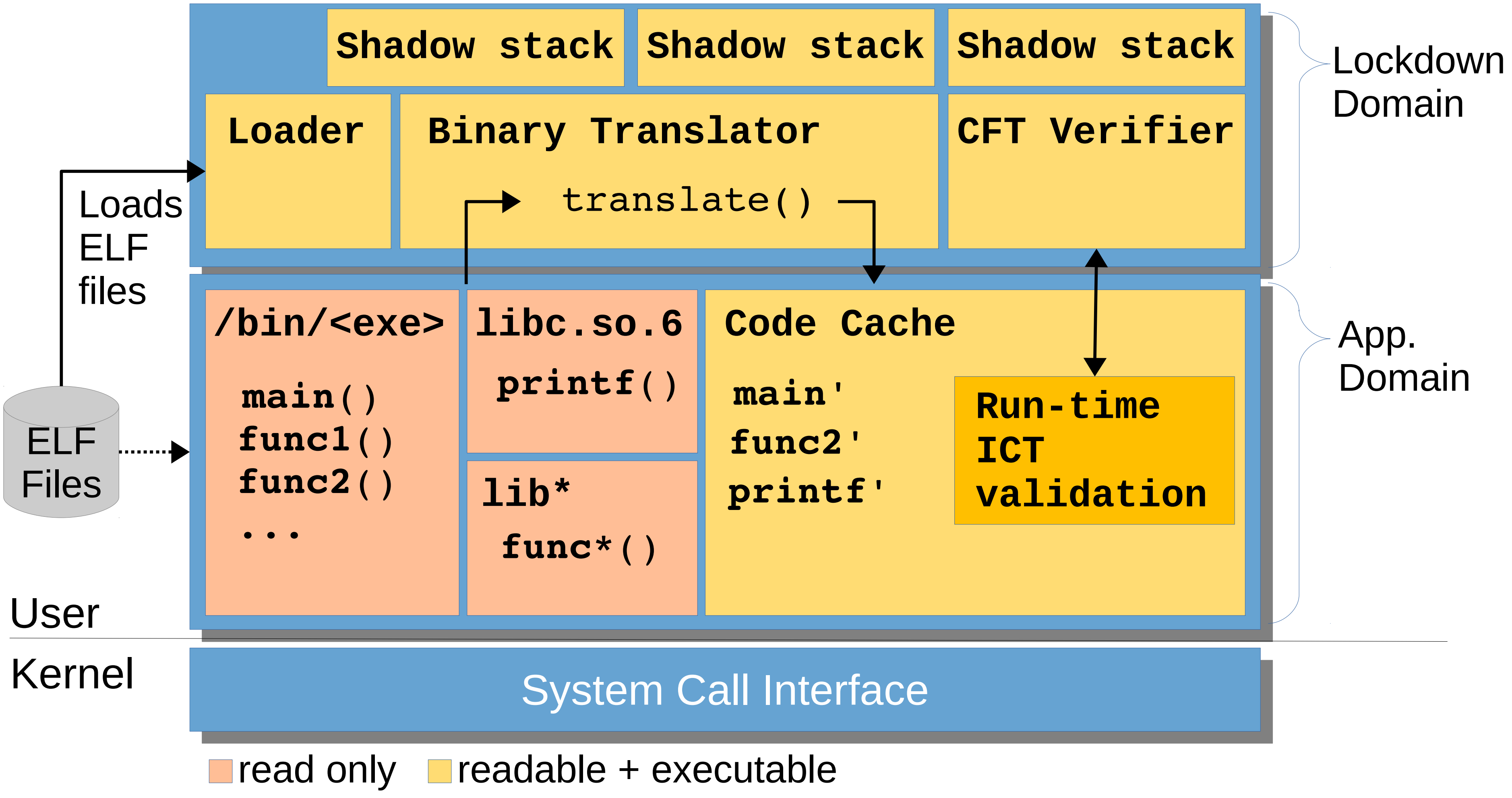
Attacker may write to memory, code pointers verified if used

Fine-grained CFI for binaries

- Fine-grained CFI relies on source code
- Coarse-grained CFI is imprecise

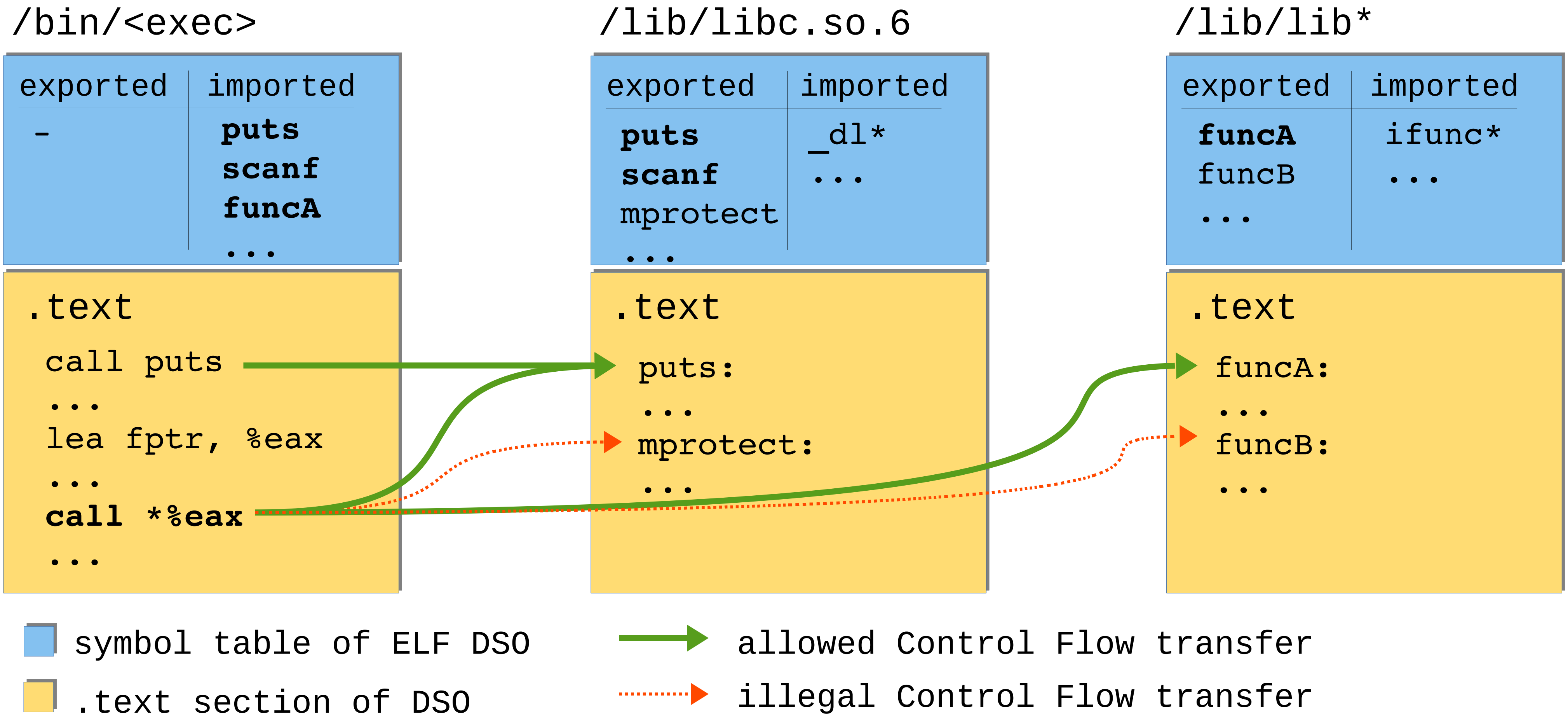
- Goal: enforce fine-grained CFI for binaries
  - Support legacy, binary code
  - Support modularity (libraries)
  - Leverage precise, dynamic analysis
  - Low performance overhead

Lockdown design



Dynamic CFI analysis

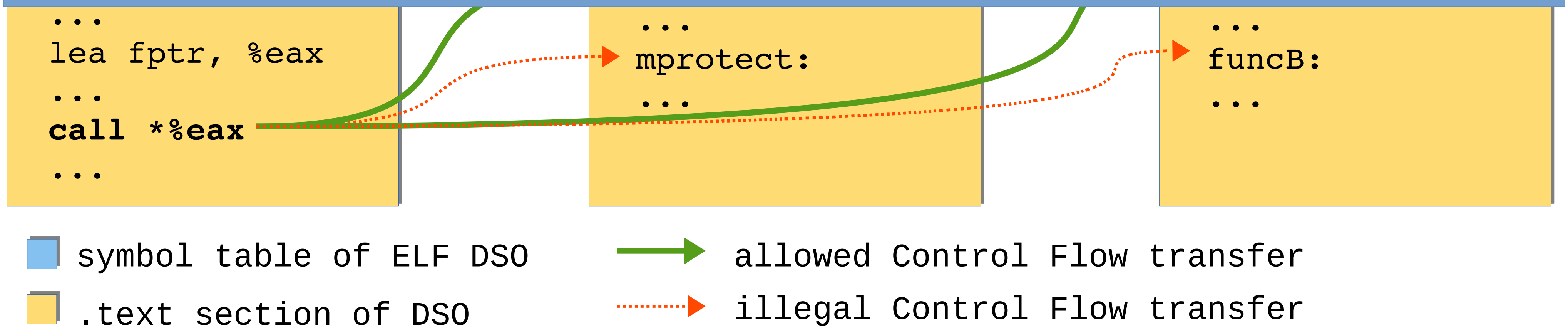
- Leverage program's modularity through loader



Dynamic CFI analysis

- Leverage program's modularity through loader

Modularity increases precision.
No source needed.
Leverage context of transfers.



```
...
lea fptr, %eax
...
call *%eax
...
```

```
...
mprotect:
...
```

```
...
funcB:
...
```

🟦 symbol table of ELF DSO

🟨 .text section of DSO

Lockdown CFI rules

- Return instructions must return to the caller
  - Precise due to shadow stack
- Call instructions must target valid functions
  - Imported in the current module (context)
- Jump instructions must target valid instructions inside the current symbol (or functions)

Performance: Apache 2.2

- 15,000,000 requests
- 56 kB HTML file, 1054 kB image
- Apache 2.2 runs under default configuration

Configuration	Small file	Image	Combined
Single threaded	30.41%	1.94%	7.87%
Concurrent	6.27%	1.09%	1.83%
Concurrent with keep-alive	15.80%	3.00%	4.36%

Security evaluation

- CVE 2013-2028 compromises nginx
  - Both ROP (ret) or COP (icall) exploitation possible

	Length	RET	CALL/JMP/ SYS
ROP attack	30	7	0
COP attack	30	0 (487*)	99

* reachable, but protected by shadow stack

Necessity of shadow stack

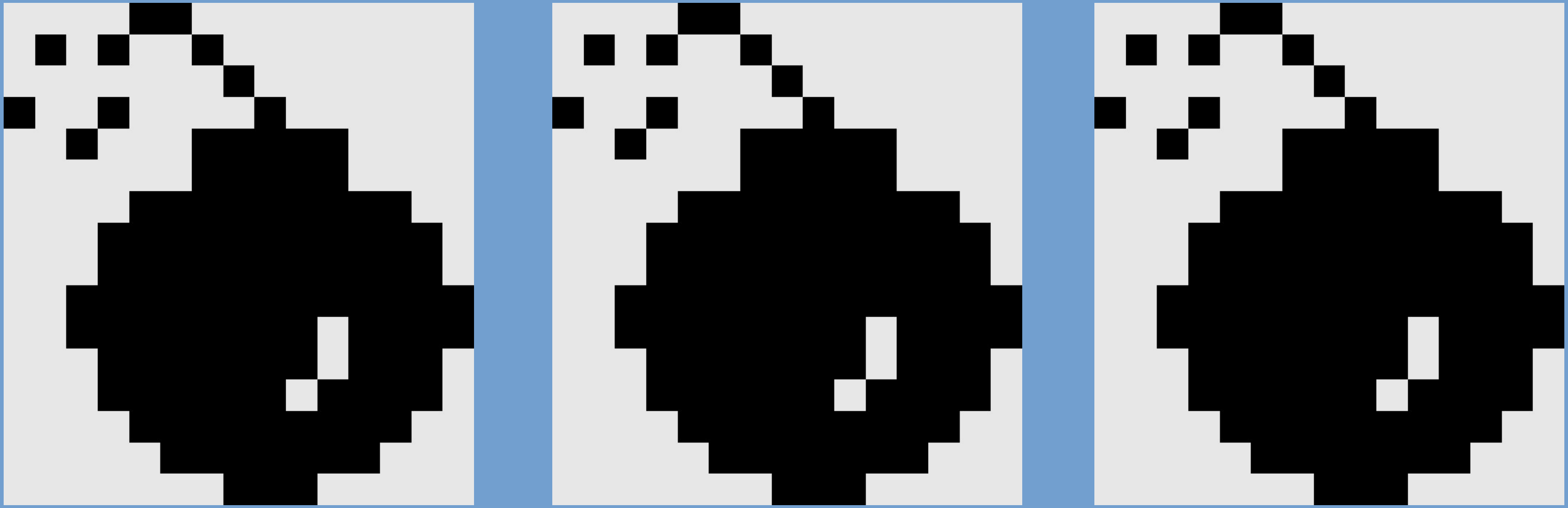
- Defenses without stack integrity are broken
  - Loop through two calls to the same function
  - Choose any caller as return location

- Lockdown enforces a protected shadow stack
  - Attacker restricted to arbitrary targets ***on*** the stack
  - Each target can only be called once, in sequence

Conclusion

Conclusion

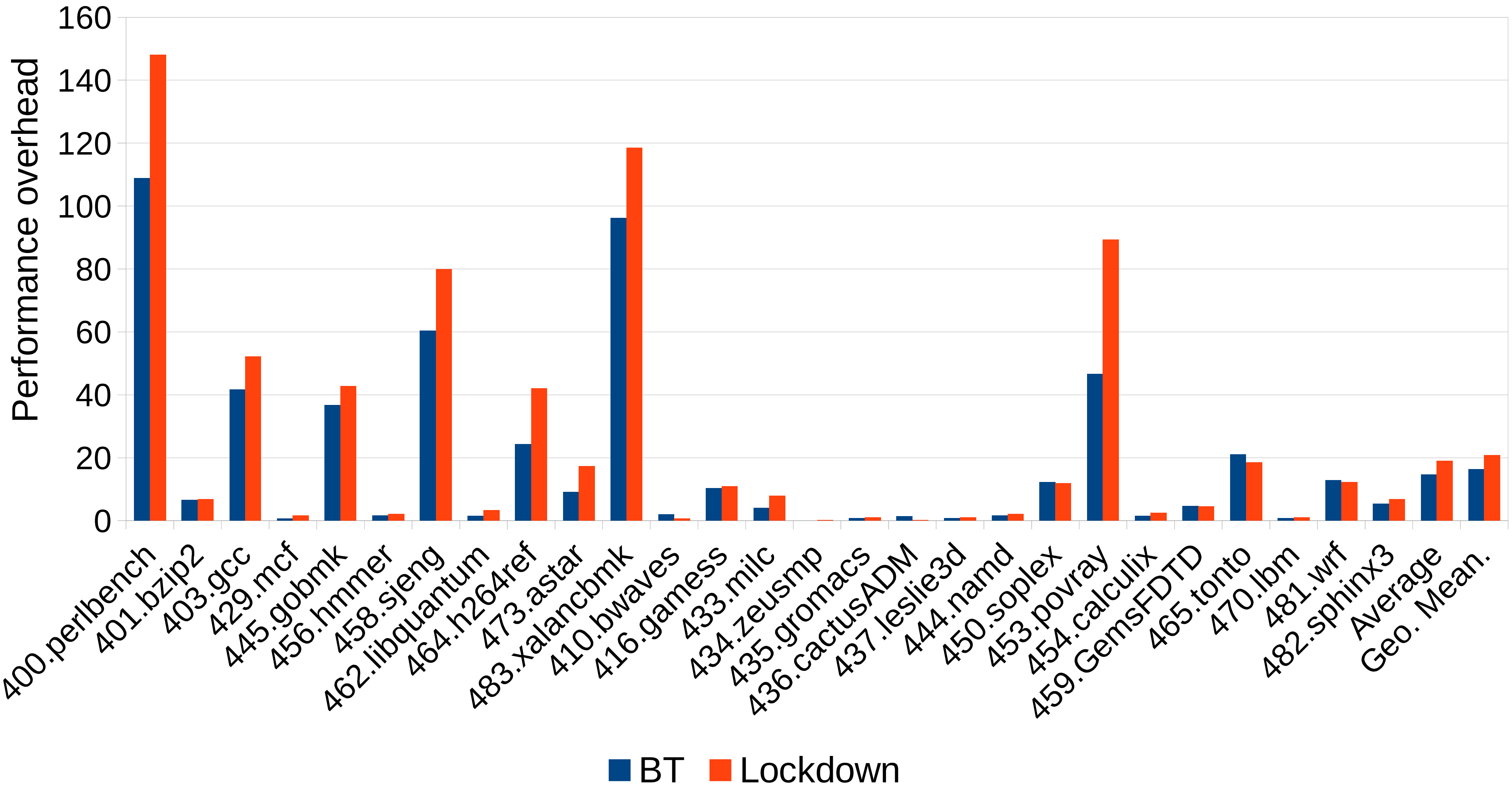
- Protect in the presence of bugs
- Supports legacy and binary code
- Control-flow hijack protection
  - Shadow stack, dynamic CFI, and locality
  - System call policy as secondary protection
- Reasonably low overhead



Thank you! Questions?

Mathias Payer, Antonio Barresi, Thomas R. Gross

Performance: SPEC CPU2006


Performance overhead
400.perlbench
401.bzip2
403.gcc
429.mcf
445.gobmk
456.hmmer
458.sjeng
462.libquantum
464.h264ref
473.astar
483.xalancbmk
410.bwaves
416.gamess
433.milc
434.zeusmp
435.gromacs
436.cactusADM
437.leslie3d
444.namd
450.soplex
453.povray
454.calculix
459.GemsFDTD
465.tonto
470.lbm
481.wrf
482.sphinx3
Average
Geo. Mean.
BT
Lockdown