Detecting Code Reuse Attacks Using Dyninst Components

Emily Jacobson, Drew Bernat, and Bill Williams

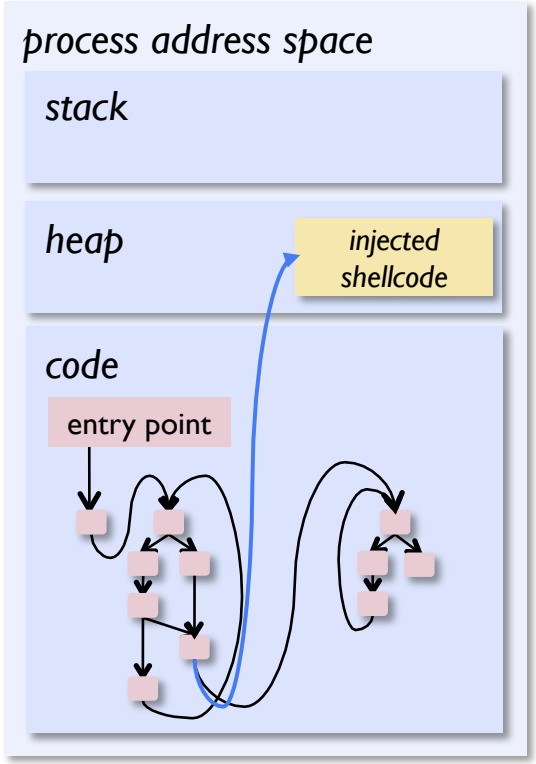
Paradyn Project

Paradyn / Dyninst Week
Madison, Wisconsin
April 29-May 1, 2013

Threat Model

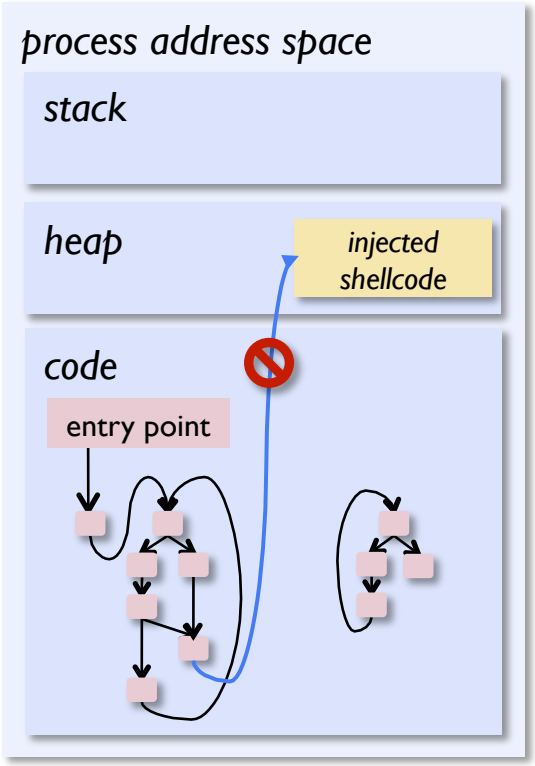
- Attack goal: effect some malicious intent by hijacking program control flow

- Historically, accomplished via code injection



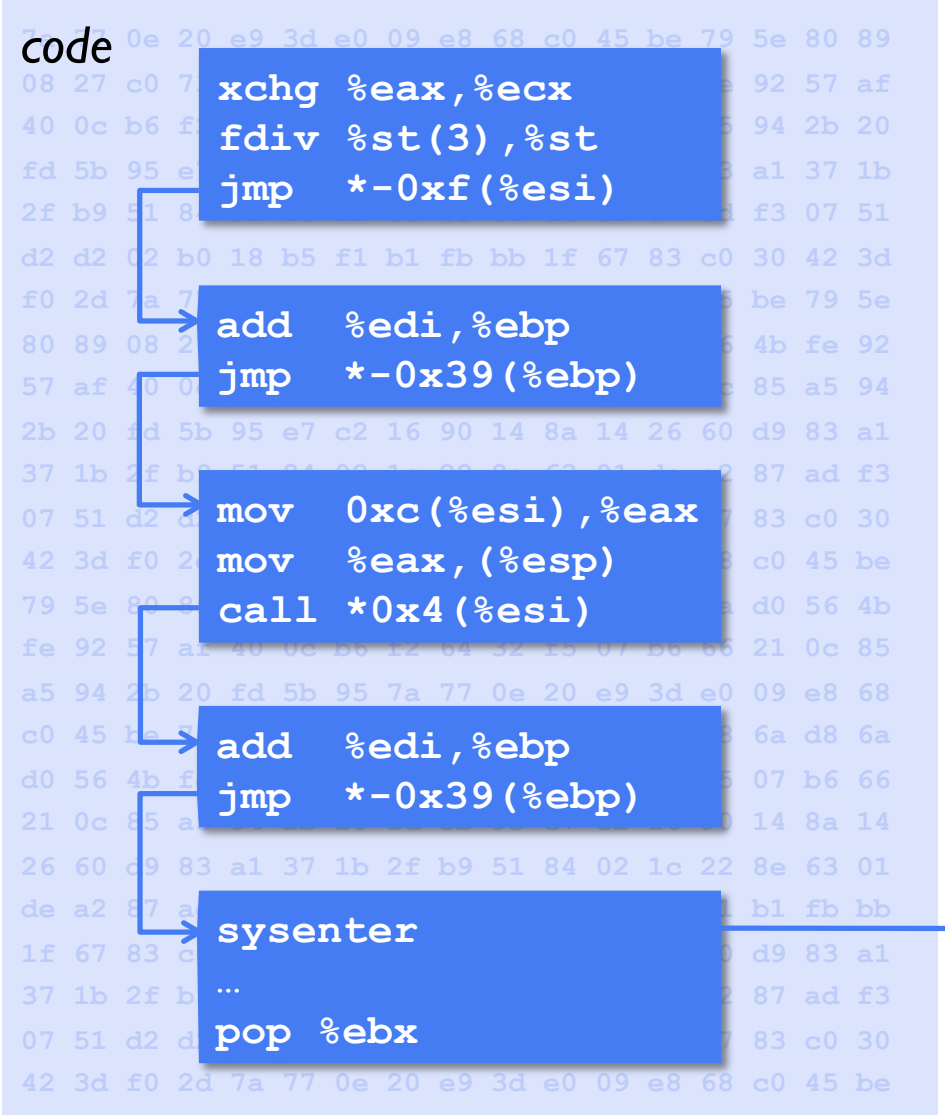
Threat Model

- W⊕X prevents code injection
- Alternative: construct an exploit using code that already exists within the program's address space



"code reuse attacks"

Anatomy of a Code Reuse Attack

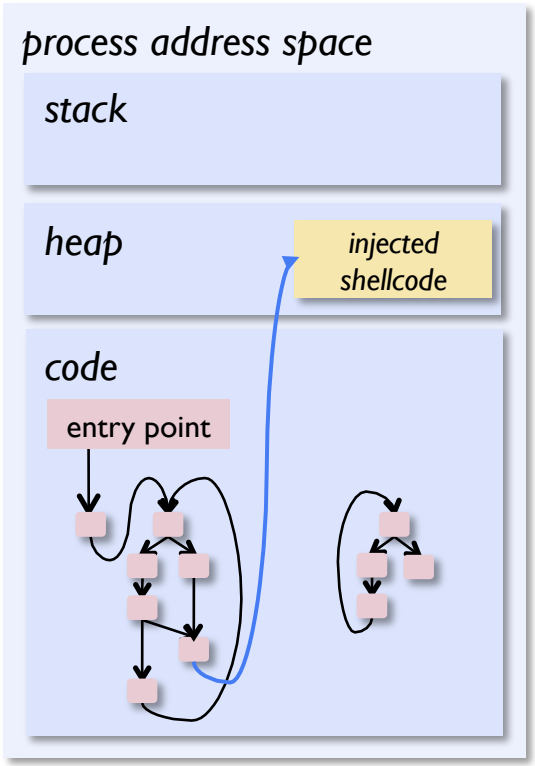


Select *gadgets* from within the address space of the process

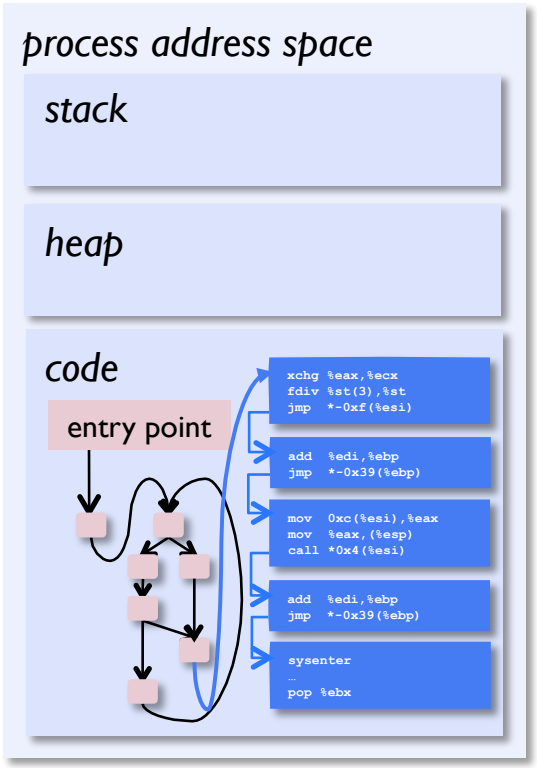
Chain gadgets together with indirect control flow

Usually a short attack with the goal of escaping the confining *W⊕X* environment

Anatomy of a Code Reuse Attack



code injection attack



code reuse attack

Previous Code Reuse Attack Defenses

- Detect using heuristics based on attack behaviors

  [Chen et al. 2009,2010], [Davi et al. 2011], [Huang et al. 2012], [Kayaalp et al. 2012]

- Enforce control flow integrity at runtime

  [Abadi et al. 2009], [Bletsch et al. 2011], [Zhang et al. 2013]

In the next talk, Tugrul will talk about another interesting defense technique.

Our Approach

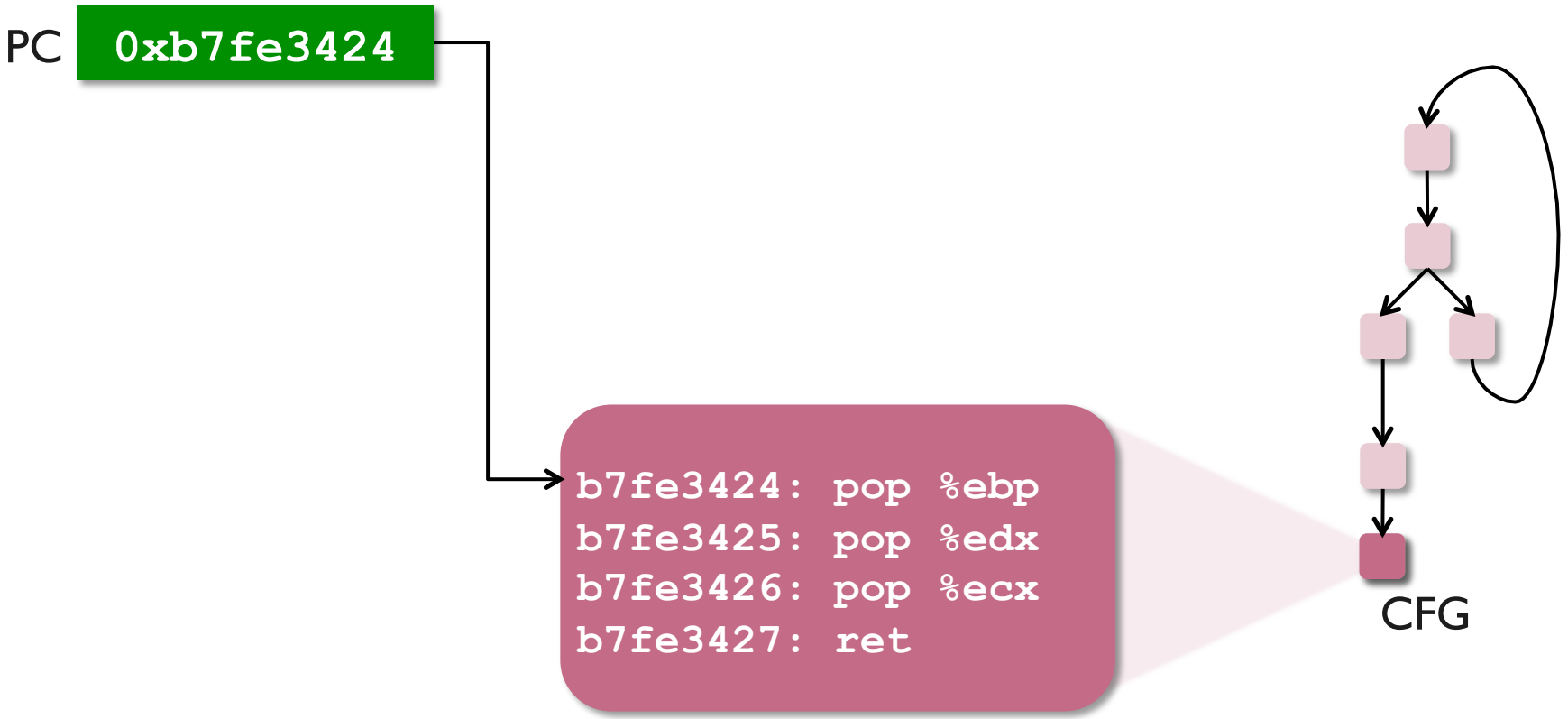
- Define *conformant program execution (*CPE*)* as a set of requirements on program states
  - Valid program counter
  - Valid callstack

- Enforce CPE by monitoring program at runtime

Model Component #1

Valid program counter (PC):

PC must point to instruction in the original program

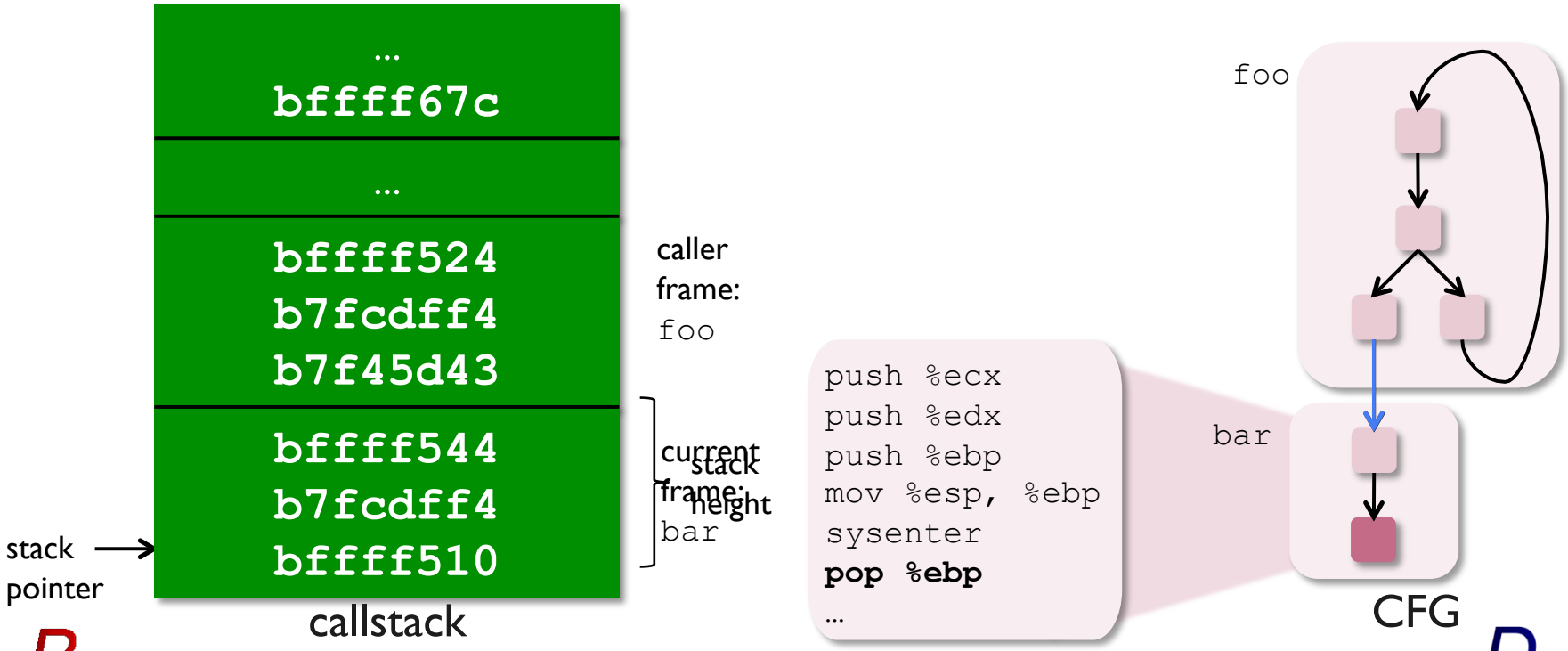


Model Component #2

Valid callstack:

For each frame:

1. *frame must have valid stack frame height*
2. *(caller → current frame) must represent a valid control flow transfer in the program*

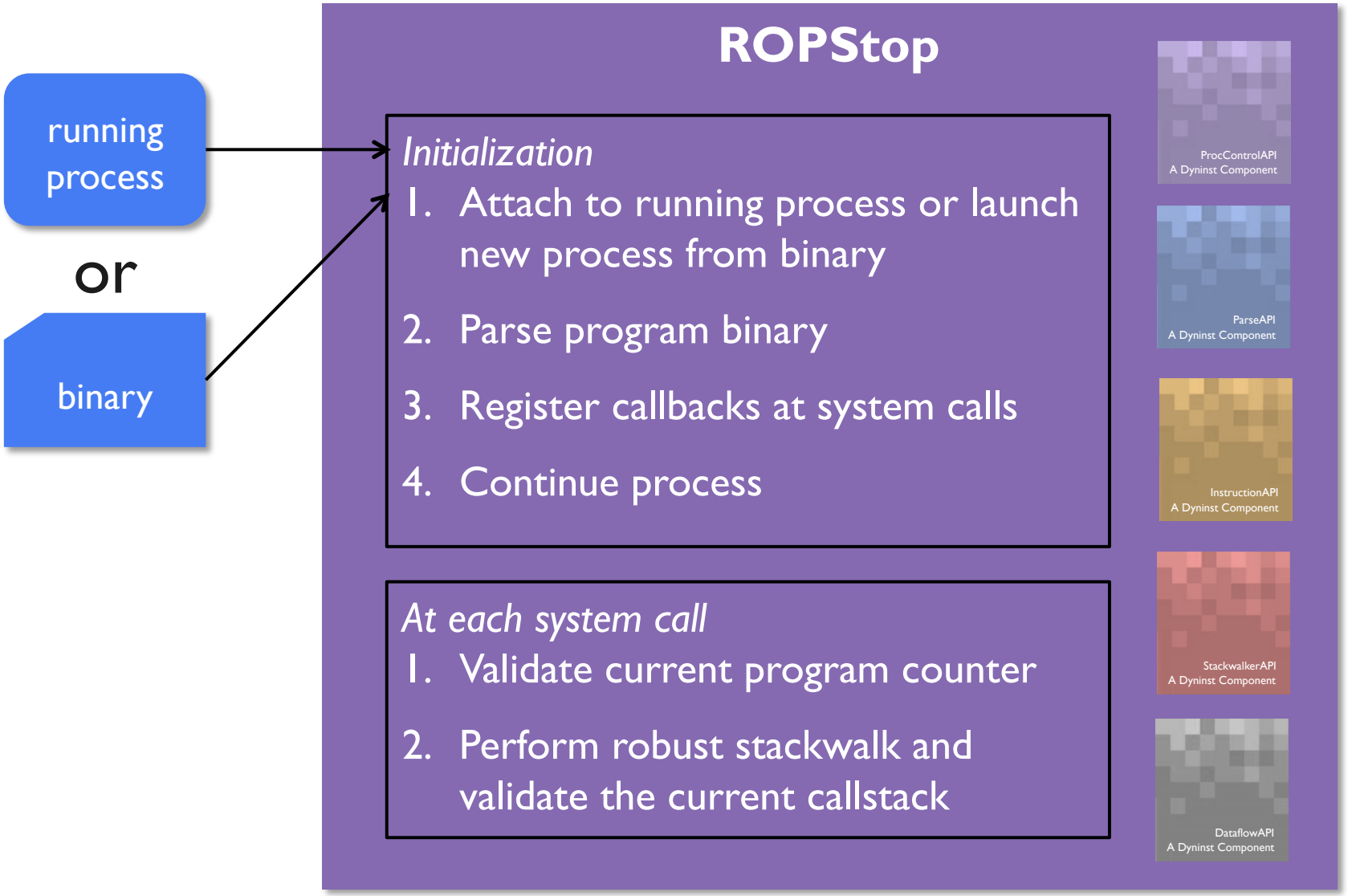


Program Validation

Design decision: when do we validate?

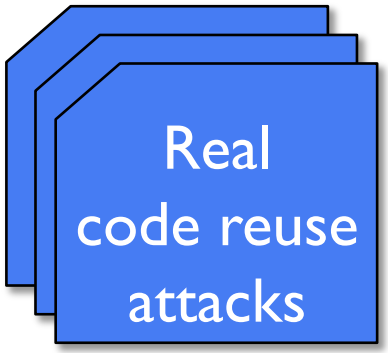
- Option 1: At all instructions
  - "Conformant program execution"
  - Disadvantage: inefficient
- Option 2: At system calls
  - "*Observed* conformant program execution"(OCPE)
  - Effective because attacks must use the system call interface to modify overall machine state

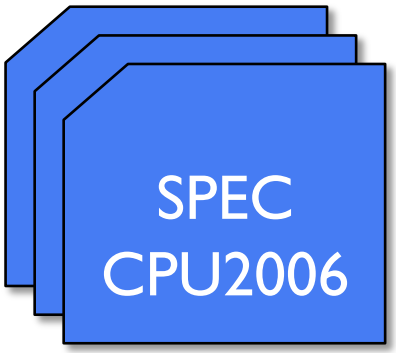
ROPStop Implementation



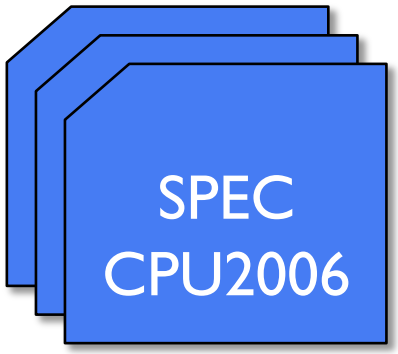
Evaluation

Accuracy





Overhead

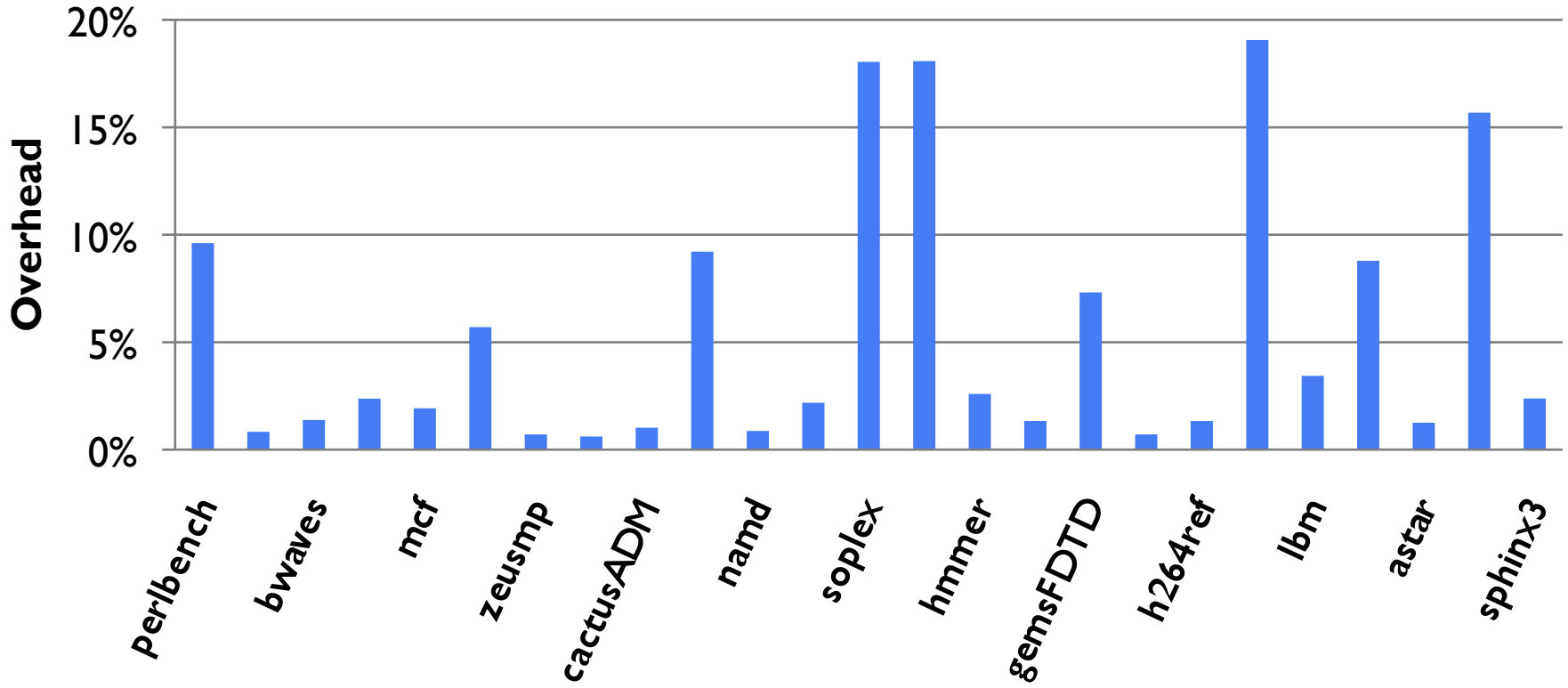


Results: Real Code Reuse Attacks

Exploit	Type	Detected	Detection Component	Why Invalid?
17286(a)	ROP	✓	Invalid stack frame height	Overwritten return address
17286(b)	ROP	✓	Invalid stack frame height	Overwritten return address
Rsync	ROP	✓	Invalid stack frame height	Overwritten return address
Bletsch	JOP	✓	Invalid stack frame height	Gadget executing
Stack-smash	Stack-smash	✓	Invalid stack frame height	Overwritten return address

100% accuracy using real ROP and JOP exploits

Results: SPEC CPU2006



100% accuracy (0 false positives),

5.42% overhead (geometric mean)

Open Questions

- Data-driven attacks (orthogonal type of attack)

  [Chen et al. 2005], [Demay et al. 2011]

- Mimicry/evasion attacks (do not exist as code reuse attacks)

  [Giffin et al. 2006], [Wagner and Soto 2002]

Conclusion

- We defined *conformant program execution* and an efficient approximation, *observed conformant program execution*

- We built a tool to enforce OCPE, ROPStop, on top of Dyninst components

Questions?

- For more details, our paper is available at:

  ftp://ftp.cs.wisc.edu/paradyn/papers/Jacobson13ROPStop.pdf

- Come see the demo on Tuesday