Binsweep

Reliably Restricting Untrusted Instruction Streams with Static Binary Analysis and Control-Flow Integrity

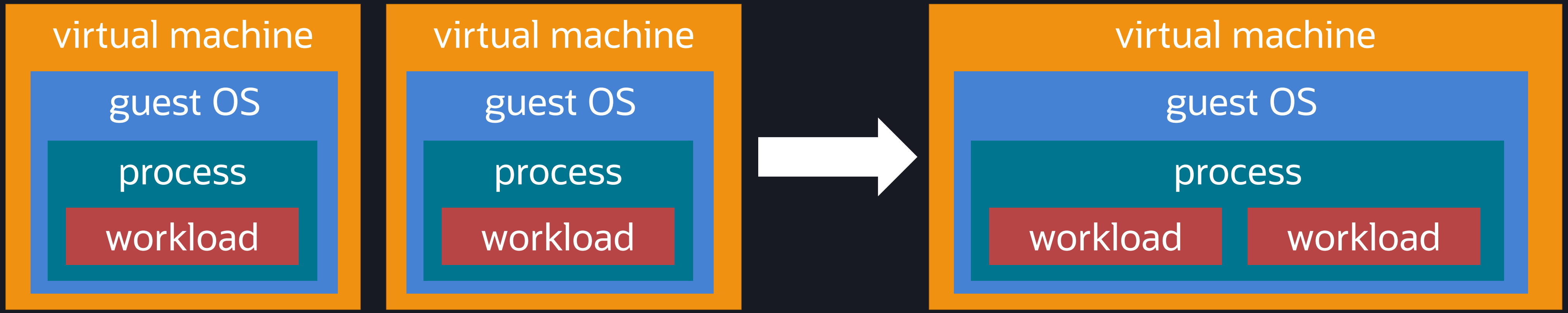
Matteo Oldani, William Blair, Lukas Stadler, Zbyněk Slachrt,
and Matthias Neugschwandtner

Oracle Labs

The ACM Cloud Computing Security Workshop (CCSW ’24)

A Trend to Lightweight Workload Isolation

- "Gold standard" of VM-based isolation incurs high overhead
  - startup, duplicated stacks, communication
  - heavyweight for small FaaS workloads

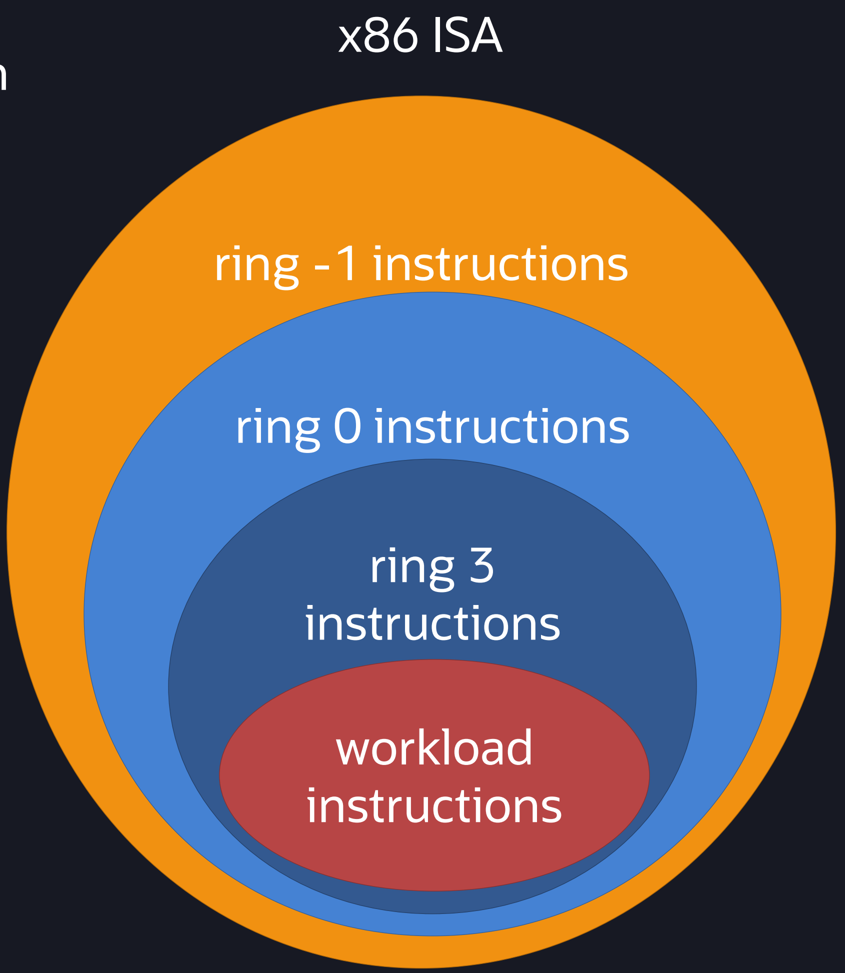


- Rekindled research on lightweight isolation
  - **ERIM**: Secure, Efficient In-process Isolation with Protection Keys (MPK), Usenix SEC'19
  - **Hodor**: Intra-Process Isolation for High-Throughput Data Plane Libraries, Usenix ATC'19
  - **Faastlane**: Accelerating Function-as-a-Service Workflows, Usenix ATC'21
  - **Jenny**: Securing Syscalls for PKU-based Memory Isolation Systems, Usenix SEC'22
  - **Endokernel**: A Thread Safe Monitor for Lightweight Subprocess Isolation, Usenix SEC'24

In-process Security Boundary

- Research leverages new hardware features for in-process isolation
  - Intel Memory Protection Keys
  - ARM Permission Overlay Extensions
- Creating an in-process security boundary introduces a new set of privileged instructions
  - changing memory access settings (`wrpkru`)
  - direct invocation of system calls (`syscall`)
  - other security relevant instructions (`clflush`)
- **PKU Pitfalls**: Attacks on PKU-based Memory Isolation Systems, Usenix SEC'20

➔ need for restricting the workloads' untrusted instruction stream



Binary Analysis

- Analyze instruction stream at the machine code level
  - no trust assumptions on how the code is generated
  - WYSIWYE (what you see is what you execute)

- Challenge: variable instruction set architecture
  - code can jump anywhere
    - inlined constants, middle of instructions
  - “ROP” gadgets hide privileged instructions

```
 0: f3 0f 1e fa          endbr64
 4: 55                   push   rbp
 5: 48 89 e5             mov    rbp,rsp
 8: 89 7d fc             mov    DWORD PTR [rbp-0x4],edi
 b: 8b 45 fc             mov    eax,DWORD PTR [rbp-0x4]
 e: 0f af c0             imul   eax,eax
11: 48 b8 f3 0f 1e fa 0f movabs rax,0xc3ef010ffa1e0ff3
18: 01 ef c3
1b: 5d                   pop    rbp
1c: c3                   ret
```

```
13: fr 0f 1e fa   endbr64
17: 0f 01 ef      wrpkru
1a: c3            ret
```

Binsweep Approach

Binary analysis challenges	Control Flow Integrity (CFI)
What are the entry points?	Links indirect branches to possible targets
What are the targets of indirect branches (`jmp`, `call`)?	Ensures that intended control flow is maintained during execution

➔ Binsweep Core Principle

assume and ensure control flow integrity during binary analysis

Binsweep Control Flow Integrity

- Requirement: ensure that all entry points and targets of indirect branches are known during binary analysis
- Precise CFG enforcement is not necessary
- CFI call/jump pattern:

callsite

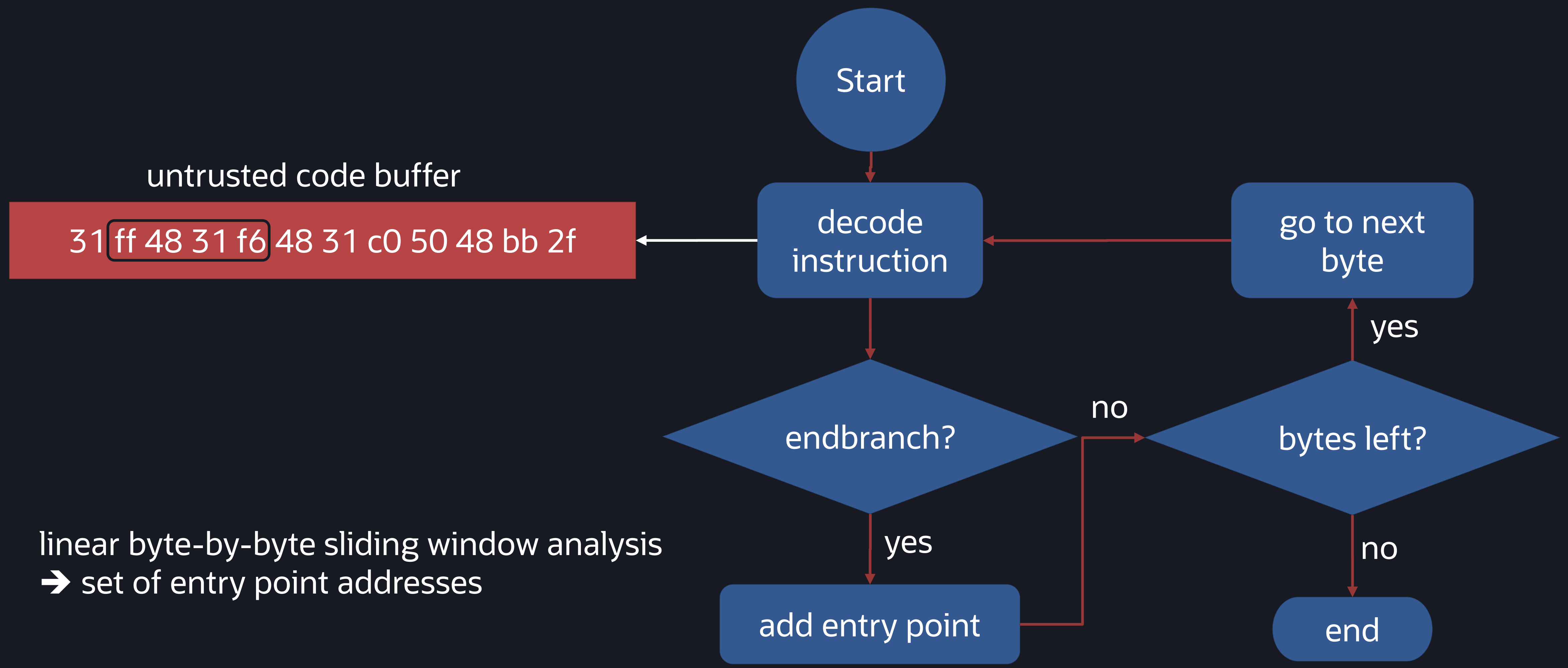
```
mov  r_target, target
mov  r_compare, [r_target]
add  r_compare, endbranch^-1
jz   succeed
halt
succeed:
call r_target
endbranch
```

call target

landing pad

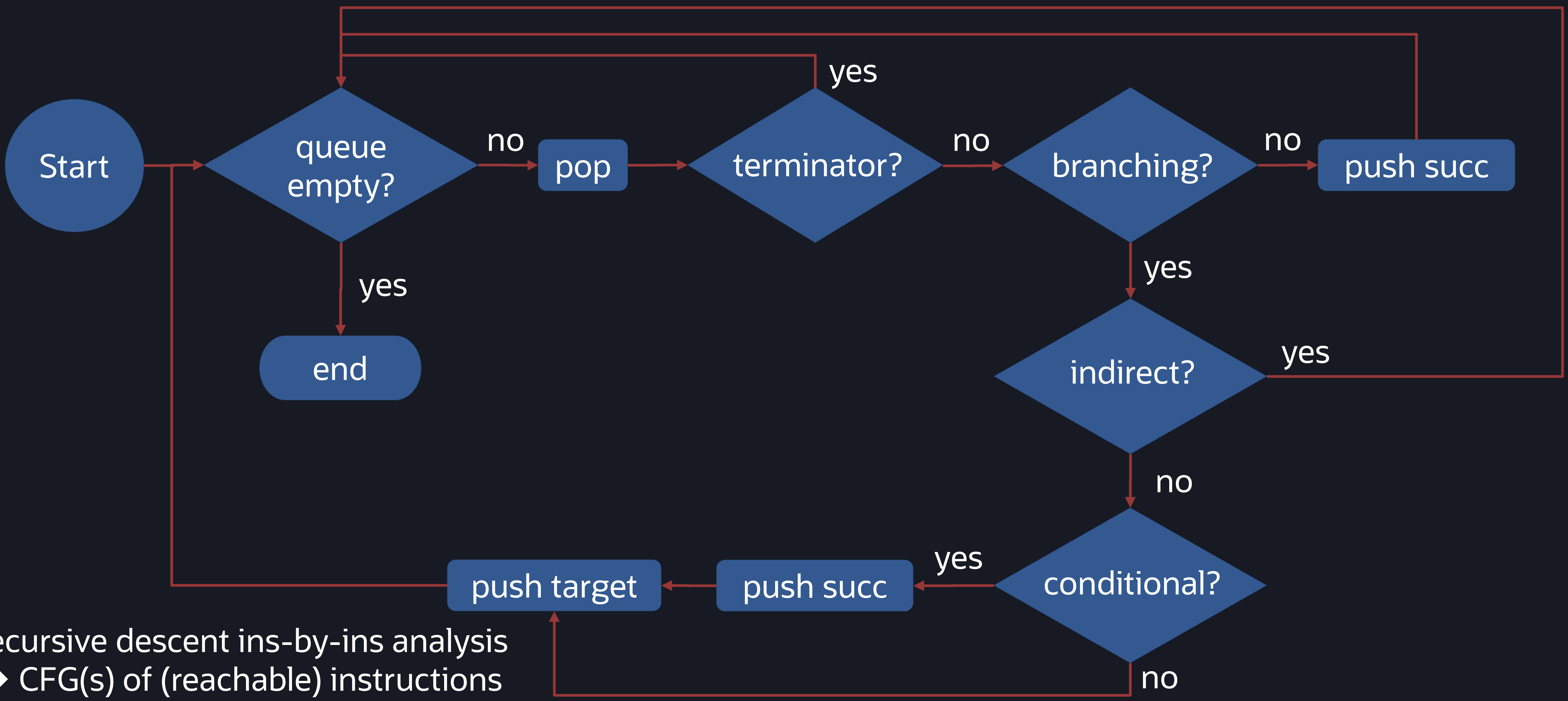
```
endbranch
…
pop  r_return
…
jmp  r_return
```

Binsweep Entry Point Detection



linear byte-by-byte sliding window analysis
➔ set of entry point addresses

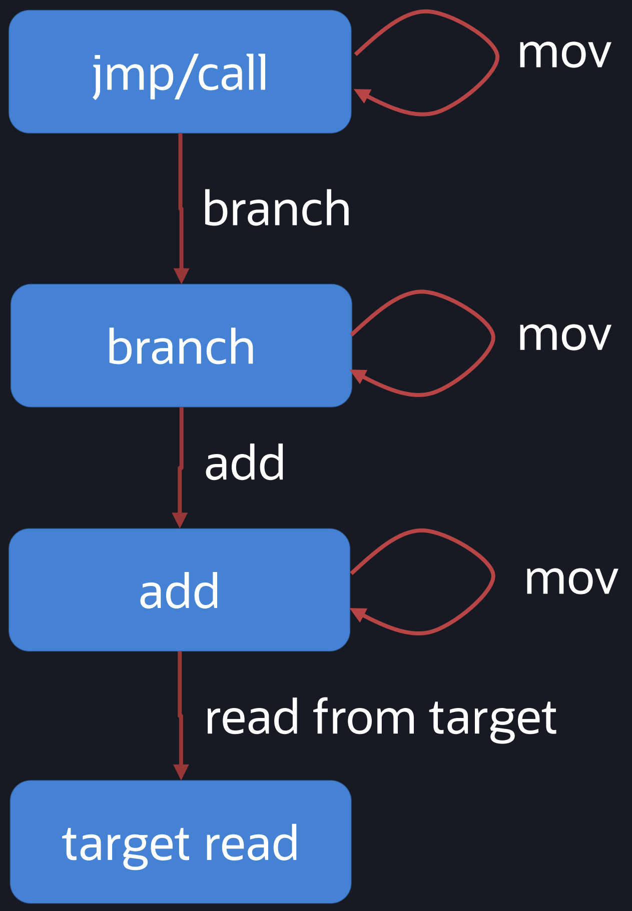
Binsweep Instruction Traversal



recursive descent ins-by-ins analysis
➔ CFG(s) of (reachable) instructions

Binsweep CFI Policy

- Enforce Control Flow Integrity
  - encode CFI pattern as an automaton
  - perform backward analysis starting at every indirect branch and apply automaton
- Challenge: interleaved instructions
  - threshold for backward analysis
- Keep track of relevant state
  - indirect branch target address register
  - branch flags register – must not be modified between add and branch operation
  - target data register – contains bytes read from the target address
- Err on the safe side

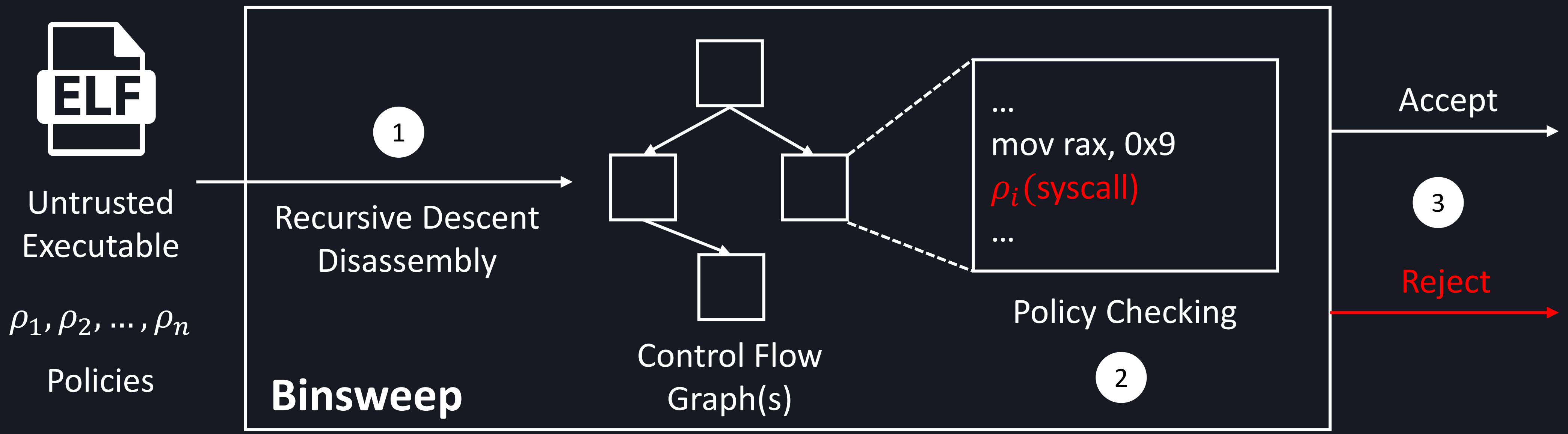


Binsweep Custom Policies

- Instruction allowlist
  - only permit instructions that are unprivileged wrt. security model e.g., do not allow `wrpkru`, `syscall`, `ret` , `rdtsc`, `clflush`

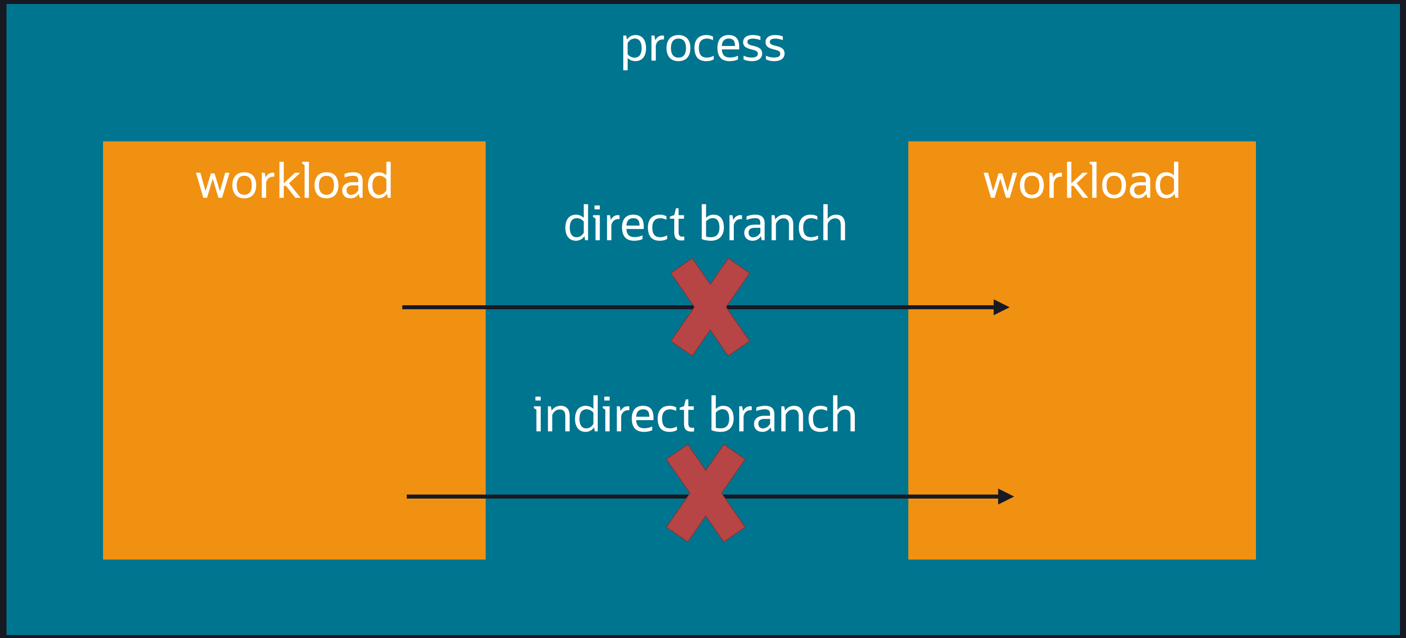
- Challenges
  - prefixes
  - implicit operands
  - mixed 64/32bit code
  - partial endbranch at a page boundary

Binsweep Overview



- “online” deployment for runtime code generation is also possible
- all code that is reachable by indirect branches has to be available at analysis time

Control Flow Isolation with Binsweep and MPK



- Binsweep enforced a data dependency on indirect branch targets
  - protection keys do not enforce instruction fetch!

- Binsweep rejects direct branch targets outside the provided code buffer

Implementation & CFI overhead

- Binsweep
  - ~10kLOC of Java code
  - compiled as a GraalVM native image
  - Intel XED as a disassembler

- Binsweep CFI
  - LLVM compiler pass, ~300LOC

SPECint 2017	CFI overhead	instrumented branches
500.perlbench	+0.02 %	4,532
502.gcc	+1.96 %	24,301
505.mcf	+1.46 %	144
520.omnetpp	+6.25 %	22,905
523.xalancbmk	+0.00 %	25,167
531.deepsjeng	+1.15 %	240
541.leela	+0.00 %	1,693
557.xz	+4.21 %	877

Performance

Program	Binary size (MB)	Binsweep verification time (s)	angr CFG recovery time (s)	basic blocks	instrumented branches
NGINX	1.3	0.087	29	25,198	3,199
CPython	16.0	0.258	116	106,095	9,261
Micronaut	77.0	4.625	-	578,047	107,787
GraalPy	362.0	18.131	-	2,474,585	384,159

https://graal.cloud

What about Intel CET?

- Hardware support for control flow integrity
  - CET IBT = Indirect Branch Tracking, forward edge, similar to binsweep CFI
    - data dependency still required if used for control flow isolation of workflows
  - CET SS = shadow stack, backward edge
    - data on the shadow stack is trusted, requires allowlist that prevents modification

- Downsides
  - only available on recent CPUs
  - Linux kernel support for userspace CET is limited to shadow stack and was only recently merged