Iodine Fast Dynamic Taint Tracking Using Rollback-free Optimistic Hybrid Analysis

Subarno Banerjee∗, David Devecsery†, Peter M. Chen∗ and Satish Narayanasamy∗
∗ University of Michigan † Georgia Institute of Technology

IEEE S&p 2019

Minkyung Park

mkpark@mmlab.snu.ac.kr

March 10, 2021

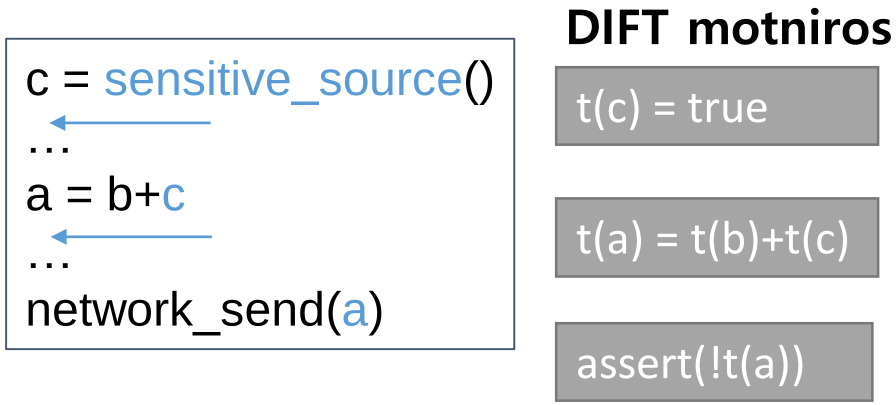
Contents

Introduction

Dynamic information-flow tracking (DIFT)

- DIFT enforces a security or privacy policy
  - Also called taint-tracking

- It tags **source** data as tainted, **propagates** taints through data and control flow, and checks if tainted data reaches **sinks**



- DIFT can help detect security attacks or prevent sensitive information from leaking through untrusted channels

Practicality

- Every instruction has to be monitored to propagate taints to the destination operand based on the source operands' taint
  - **Prohibitive performance overhead**
  - Slowdown up to 1-2 orders of magnitude

- How to reduce this cost
  - Reducing tainted sources
  - Coarsening the granularity of objects
  - Parallelizing
  - …

→ Compromise accuracy

→ Throughput overhead

Optimistic hybrid analysis (OHA)

- Execution paths that violate an information-flow policy are almost either **rare or impossible**
  - DIFT fundamentally do more work than necessary

- OHA uses both static analysis and dynamic analysis to **elide *likely* unnecessary DIFT monitors**

- A static analysis can identify these instructions and elide DIFT monitors for that

- The soundness problem: the elided instructions may be necessary monitors

  → the program execution is replayed from the **beginning**

Iodine

- A novel OHA approach that enable efficient and sound DIFT for live execution

- Iodine eliminates the need for rollback and enables forward recovery

- Any monitor elided during a program execution has to be proven to be unnecessary to ensure soundness → safe elision

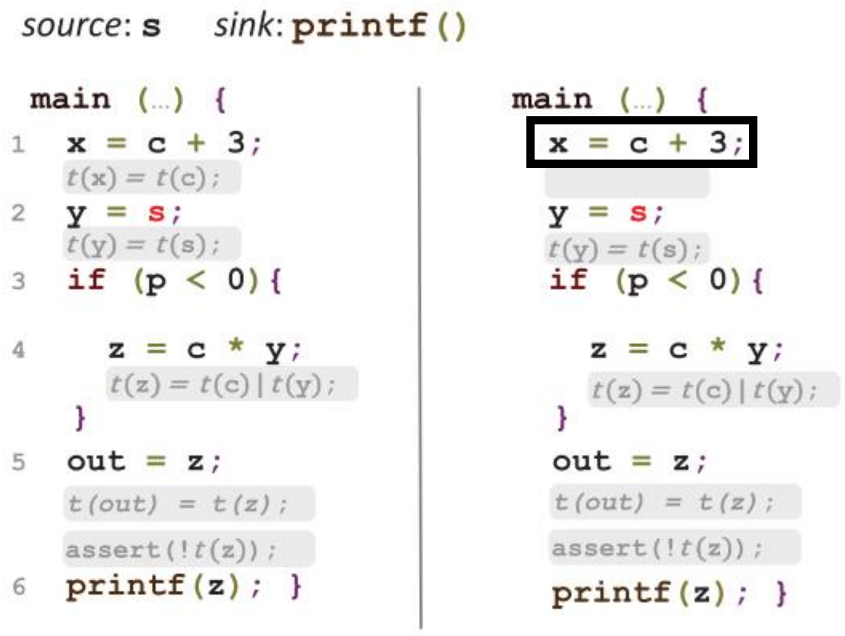
Background

Conservative hybrid analysis

- A pure DIFT instruments all instructions to propagate taints
- Information-flow leaks are rare
  - Not propagating taints or not reaching any sink
- The hybrid analysis optimizes its dynamic taint analysis
  - Static analysis can be used to remove unnecessary monitors
- There are two ways in the hybrid analysis
  - Forward taint analysis
  - Backward taint analysis

Forward taint analysis

- It determines if the source operands of an instruction may be tainted
- If none of the source operands may be tainted, then its track monitor is pruned



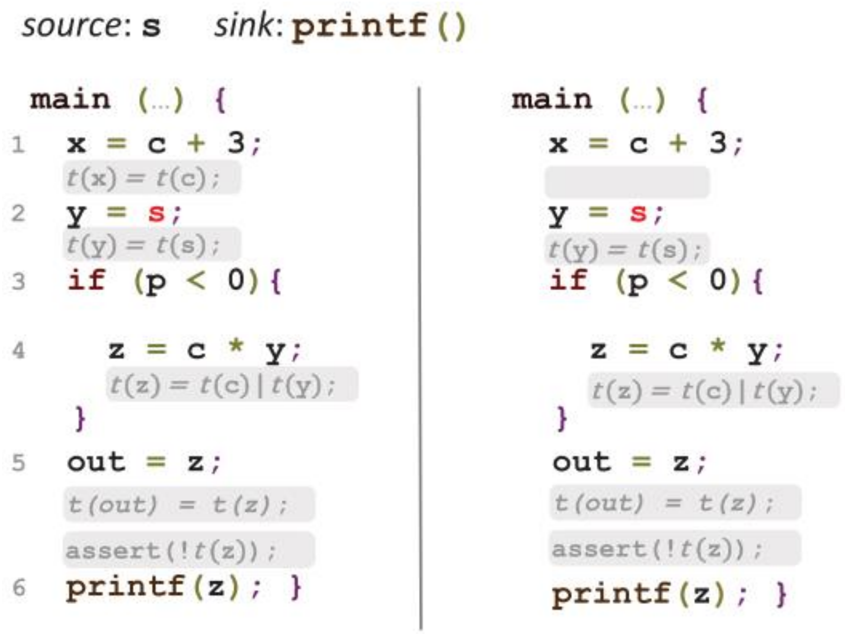
(a) Full dynamic analysis (b) Conservative hybrid analysis

← Neither source operands are tainted
x will not be tainted

Backward taint analysis

- It determines whether a destination operand of an instruction may reach a sink

- If not, track monitor for that instruction is elided (even if it can be tainted)



(a) Full dynamic analysis (b) Conservative hybrid analysis

← Cannot leverage this property soundly

Optimistic hybrid analysis (OHA)

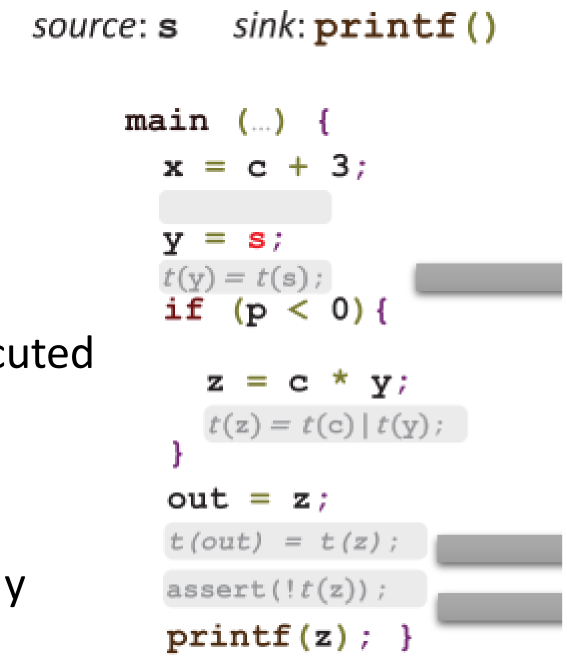
- Conservative hybrid analysis is still limited
  - Many infeasible program states is included
  - Most executions cover only a small subset of common execution states
- OHA consider the states that will be realized in the dynamic executions
- An OHA profiler observes representative executions to gather ***likely invariants***
  - e.g., unreachable code, callee sets, unrealized call contexts
  - These are mostly true, but are hard to prove statically
- The likely invariants are used as predicates for forward & backward analysis
  - Resulting in a <u>predicated</u> static taint analysis

Example of OHA

- The executions only have **”p>=0”**

→ **“z=c*y”** is never executed

→ The variable z
does not tainted due to y



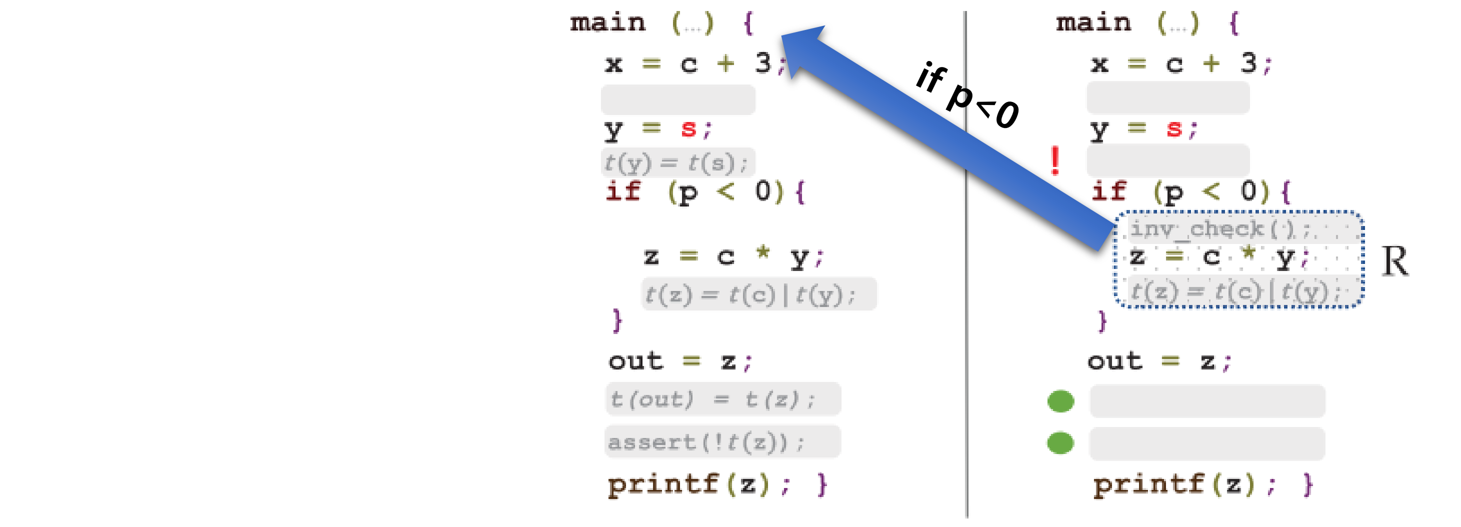
(b) Conservative hybrid analysis

ıitor: never reach sink

forward monitor: source operand never tainted

Problem: rollback recovery in OHA

- When a likely invariant fails, the predicated static analysis is rendered as ***unsound***
- When it fails, the program execution is replayed from the beginning using the conservative hybrid analysis



(b) Conservative hybrid analysis (c) Optimistic hybrid analysis

- A rollback to the beginning compromises availability of the system

Iodine

Safe elisions

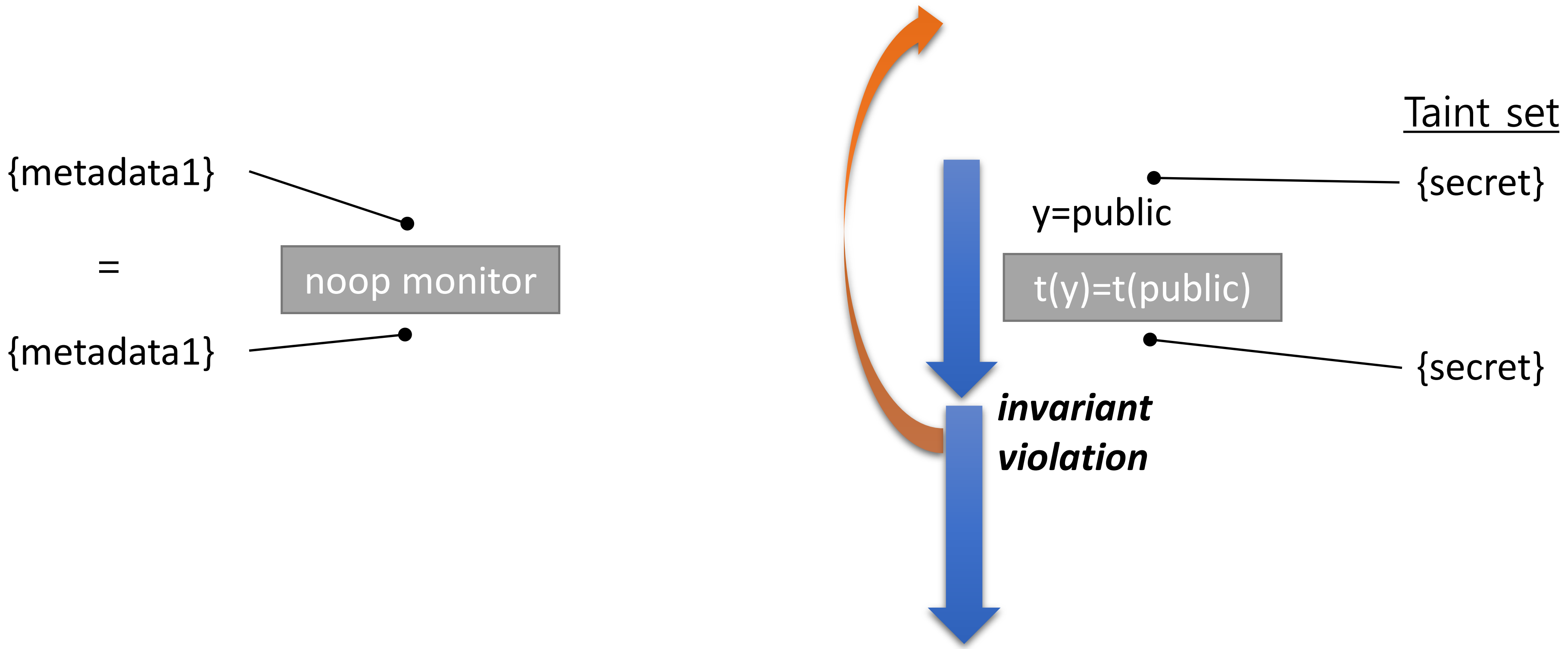
- Iodine is a **rollback-free** OHA using **safe elision**
  - The need for rollback on invariant failure is eliminated

- Rollbacks are cased by the dependence between the current monitor being elided and potential future invariant failures

- Iodine elides a monitor when it can prove that an invariant violation would not affect any preceding elisions of that monitor

Noop monitor elisions

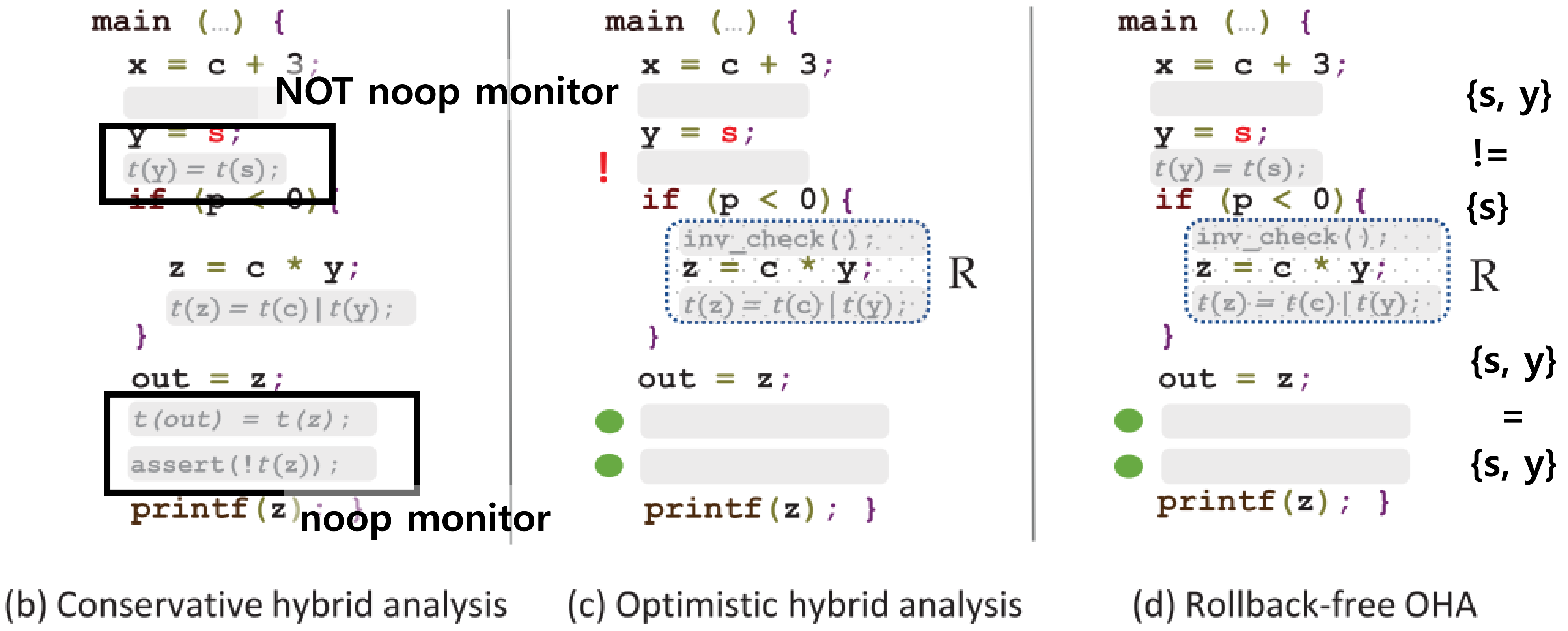
- A noop monitor is one that does not change the analysis metadata state



- Elisions of noop monitors are safe elisions

Noop monitor elisions

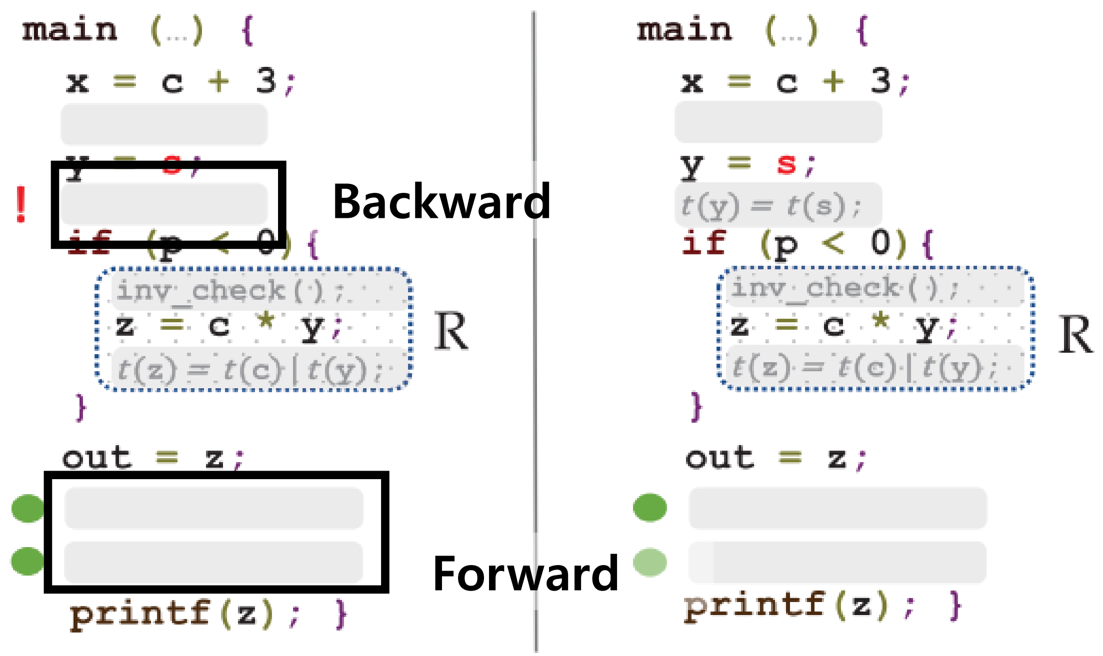
- We assume R is unreacahble



source: `s`   *sink*: `printf()`

Noop monitor elisions

- Predicated forward optimizations are safe
  - All elided monitors are noop monitors
- Predicated backward optimizations may not be safe

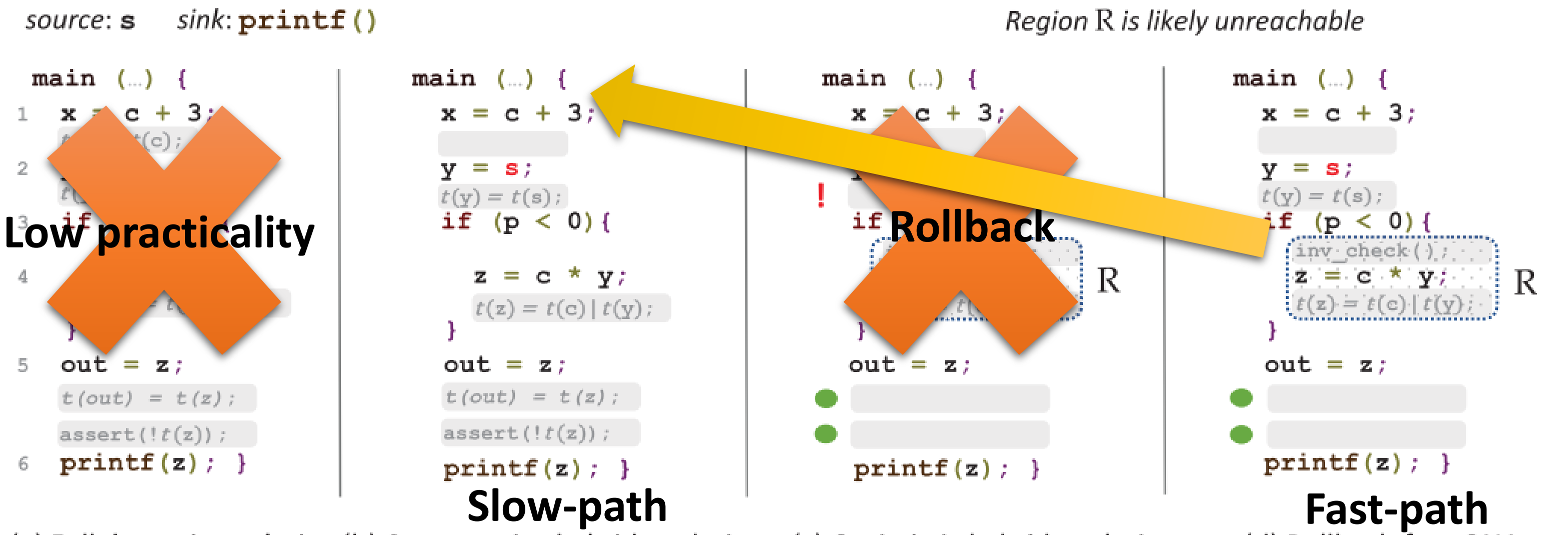


(c) Optimistic hybrid analysis (d) Rollback-free OHA

Rollback-Free Optimistic Hybrid Taint Analysis

- Iodine uses predicated forward analysis and conservative backward analysis

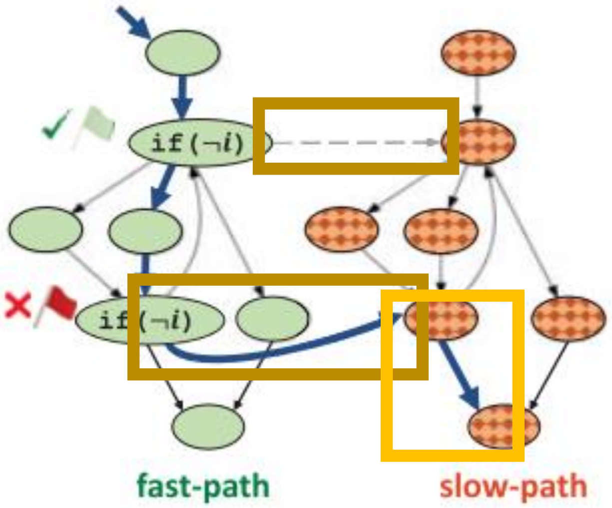
- How to treat invariant violation
  - It instruments a conditional branch for every invariant check
  - **Optimized dynamic analysis (fast-path)** is executed until an invariant fails
  - The invariant check switches the control to **a conservatively optimized analysis (slow-path)**



(a) Full dynamic analysis (b) Conservative hybrid analysis (c) Optimistic hybrid analysis (d) Rollback-free OHA

Forward recovery mechanism

- Each function implements both the fast-path and the slow-path code
  - The control flow graph for a function is replicated
- A conditional jump to the slow-path is inserted to each invariant check
  - When invariant fails, the execution is switched
- All functions in the call stack must switch to the slow-path upon a return from the slow-path domain
  - After every call site, a conditional switch switches to the slow-path



Evaluation

Experimental setup

- Implementation: LLVM compiler infrastructure supporting C language
  - LLVM's Data Flow Sanitizer as instrumentation backend
- Environment: a single core of an Intel Xeon E5-2620 processor with 16GB RAM
- Benchmark suit
  - Postfix mail server test generators
  - nginx/thttpd: serving webpages
  - redis: database server
  - vim: text processing
  - gzip: (de-)compressing files
- Profiling executions to gather likely invariants
  - Postfix stress tests
  - ngnix, thttpd serving pydoc3 documentation and loading webpages
  - redis benchmarking application and performing geo-search
  - vim challenge solutions
  - gzip with SPEC's bzip2 and sphinx reference inputs

→ A profile set of 400 executions, and a performance test set of 100 executions

Iodine framework overhead

- Invariant check overhead
  - Invariant checks have nearly no effect on runtime, incurring only 2% of overall execution time

- Invariant violation overhead
  - During some-to-all analysis, only sendmail, redis and vim violates an invariant in 3, 2, and 5 (out of 100) executions respectively
  - The amortized overhead of the slow path analysis resulting from the invariant violation is less than 0.5%

IFT Security policies

- Security policy from Dytan (related work) and Google desktop's privacy policy
  - Email integrity and privacy: receiver addresses are entirely determined by user input and message dates are only determined by the time syscall, etc.
  - Overwrite attacks on web server: taints all network inputs, and asserts that tainted values are not used as function pointers, etc.

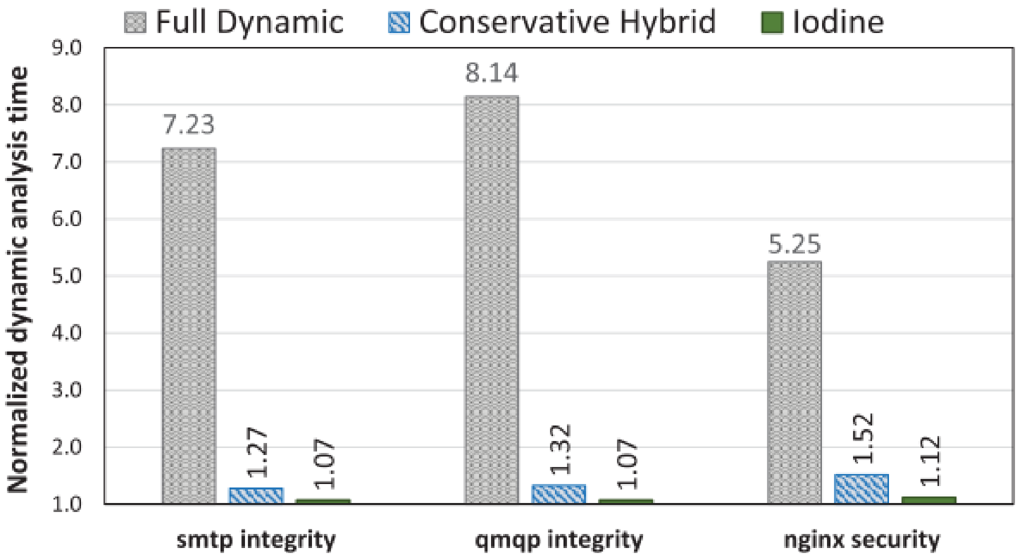
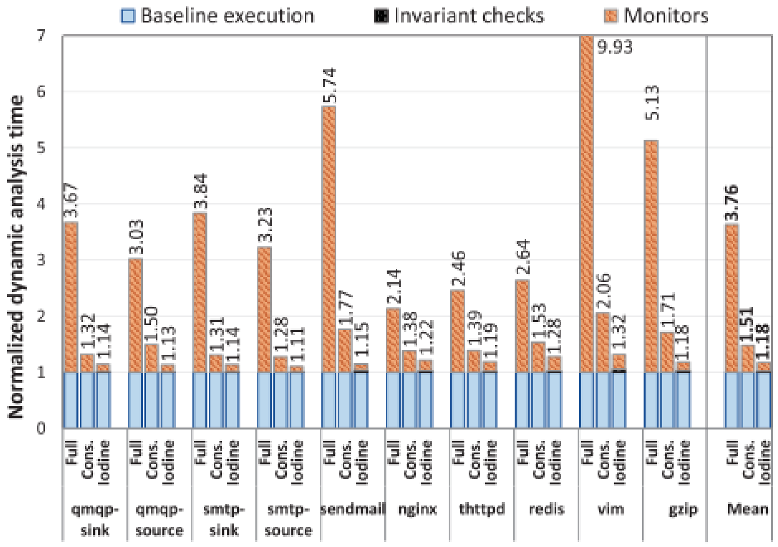


Fig. 5: Dynamic information-flow tracking applications

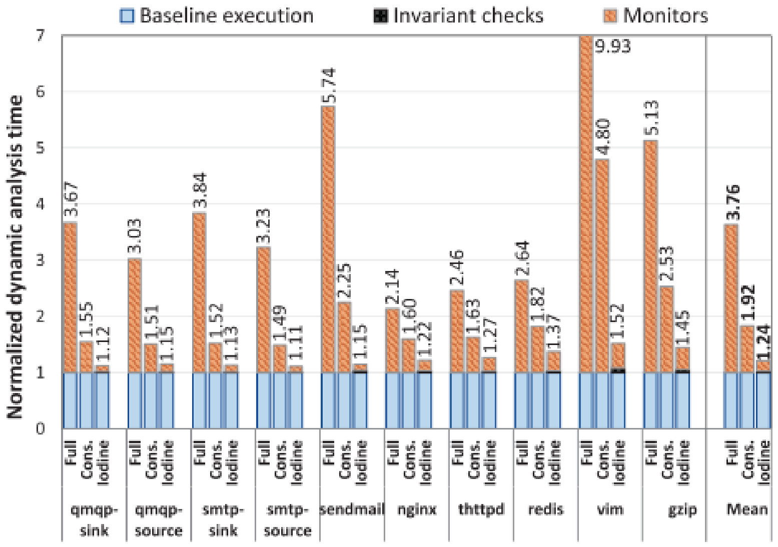
The effectiveness of Iodine using real taint policies
→ 4.4x reduction in runtime overhead

Generic information-flow policies

- Two different variants of taint analysis is implemented to evaluate the effectiveness of Iodine in a forward-only analysis vs. a forward-backward analysis
  - Some-to-some: propagates taint from a randomly sampled fraction of the taint sources to the set of all sink instructions → both forward and backward analyses are used
  - Some-to-all: treats all instructions as potential sinks and propagates taints from the sampled taint sources → only forward analysis is used



(a) some-to-some taint analysis



(b) some-to-all taint analysis

→ Iodine significantly reduces the runtime overhead

Conclusion

Conclusions

- Optimistic hybrid analysis (OHA) to optimize dynamic information flow tracking (DIFT) suffers from **rollback recovery problem**
- Iodine presents a novel approach by eliminating the need for rollbacks
- Iodine restricts predicated static analysis optimizations to **noop safe elision**
- Thereby, it improves the precision of static analysis and reduces runtime overhead