RAIN: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking

Y. Ji, S. Lee, E. Downing, et.al.

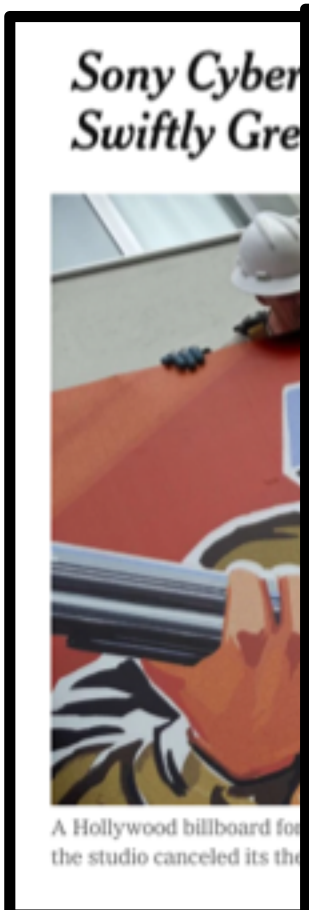
CCS'17

Presented by: Mohammad A. Noureddine

CS563

Fall 2018

No Shortage of Recent Breaches!



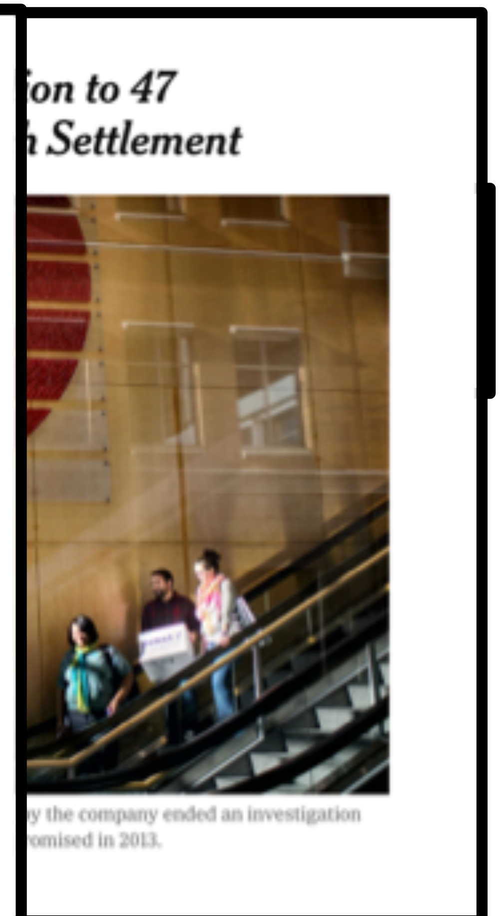
Sony Cyber
Swiftly Gre

A Hollywood billboard fo
the studio canceled its th

Facebook Security Breach Exposes Accounts of 50 Million Users

One of the challenges for Facebook's chief executive Mark Zuckerberg is convincing users that the company handles their data responsibly.
Josh Edelson/Agence France-Presse — Getty Images

By Mike Isaac and Sheera Frenkel



ion to 47
h Settlement

y the company ended an investigation
omised in 2013.

Investigating Attacks

- Definition: *Whole-system provenance*
  - *"A complete description of agents (users, groups) controlling activities (processes) interacting with controlled data types during system execution"* [1]

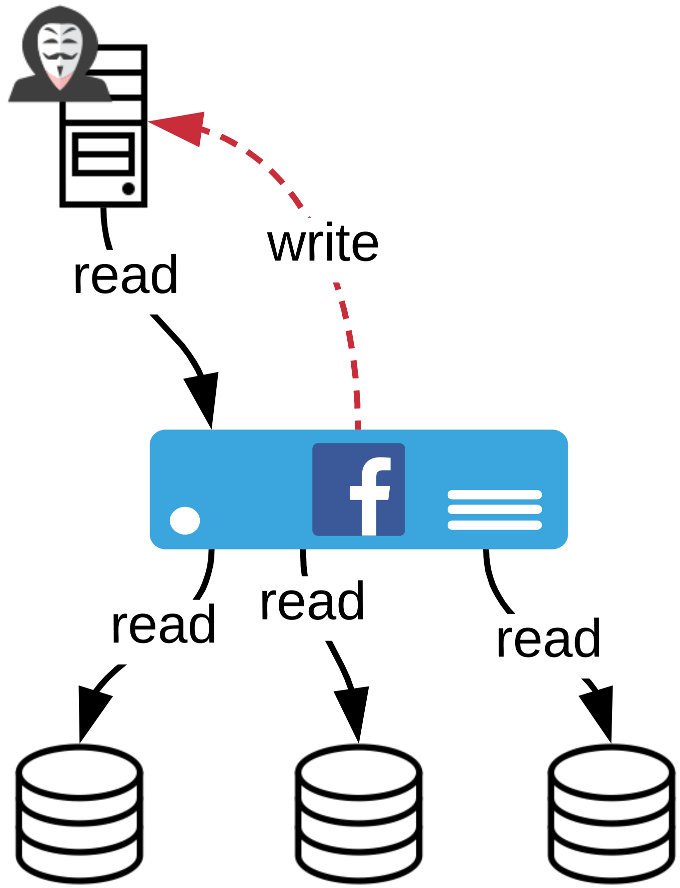
- Determine the root cause of a breach
- Determine the impacts of an exploit on the system

[1] Bates, Adam M., et al. "Trustworthy Whole-System Provenance for the Linux Kernel." *USENIX Security Symposium*. 2015.

Provenance Graphs

- Track and Log system Interactions
  - Usually system-call level

- From a given point of interest
  - Can determine root cause
    - Backward traversal
  - Can determine impact on the system
    - Forward traversal



Provenance Graphs: Challenges

“Dependence Explosion” Problem

Traditional Approaches

- Tradeoff performance vs graph granularity
- System-call tracing
  - Better performance but not enough granularity
- Dynamic Information Flow Tracking (DIFT)
  - Fancy name for taint analysis
  - Better granularity but worse performance
- DIFT + record and replay
  - Performance hit becomes someone else's problem

This Paper

- RAIN: Refinable Attack INvestigation
- Combine best of each approach!
  - System-call level graph generation
  - Graph pruning
  - Record & Replay
  - Selective DIFT

Good Runtime Performance

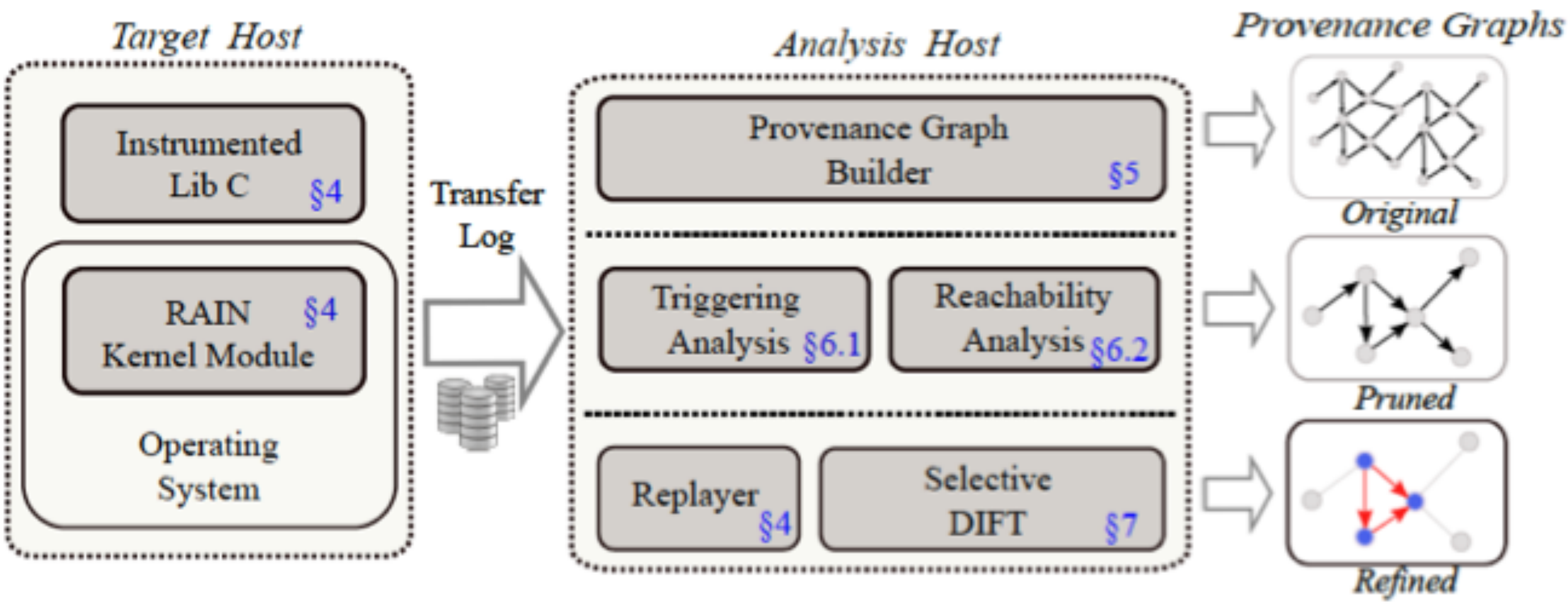
Reduce performance hit of DIFT

Improved granularity!

What Can the Attacker Do?

- Kernel: Good
  - *Kernel* and *monitoring system* form a *trusted computing base (TCB)*
- User space: Bad
- No side channels

High Level Overview



Logging Behavior

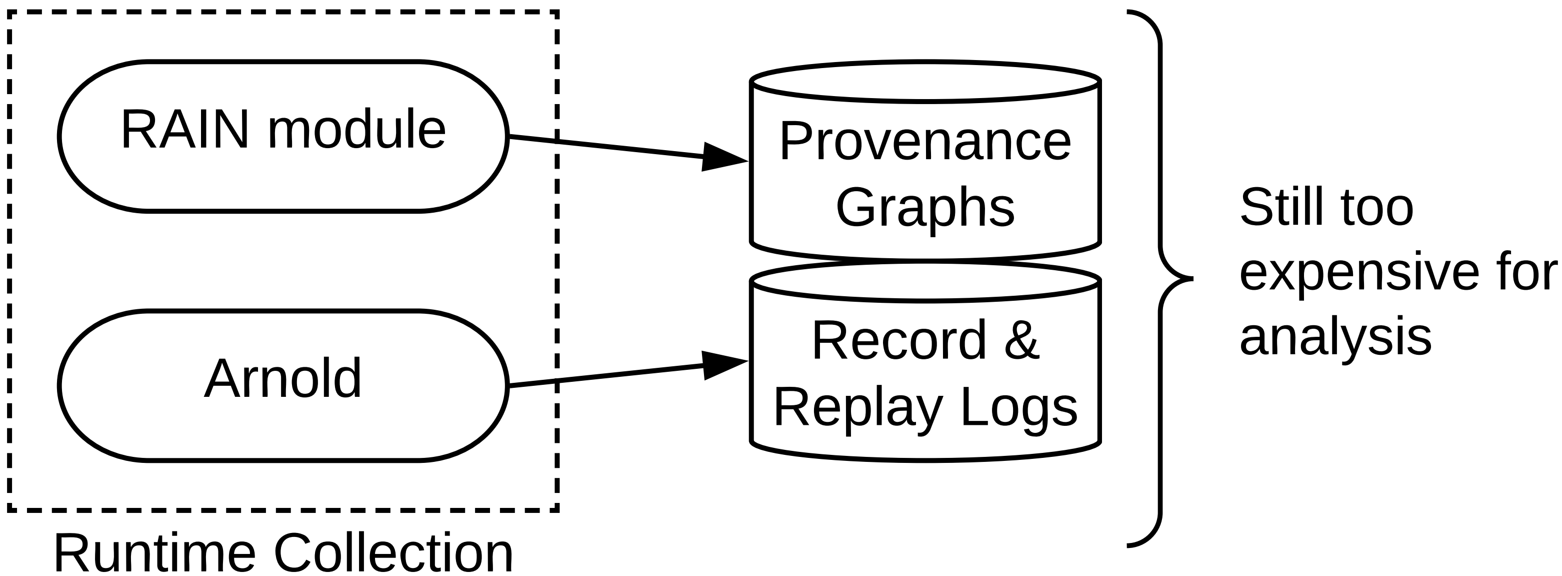
- Logging component resides completely in the kernel
  - Trusted given the threat model of the paper
  - Capture system calls, their arguments, and return values
  - `read, write, open, send, recv, connect`
  - Build the same traditional provenance graphs

- Keep logs not only to infer causality
  - Need to be able to *faithfully replay* the system's execution

Record & Replay: Arnold

- Capture non-determinism for later replay
- Goal is to reproduce complete architectural state of a user process
  - Record IPC communications
  - Cache data of every file and network I/O
- Record non-determinism by instrumenting `pthread` in `libc`
  - Enforce determinism when replaying

Story so far



RAIN module
Arnold
Runtime Collection
Provenance Graphs
Record & Replay Logs
Still too expensive for analysis

PRUNING I: Triggering Points

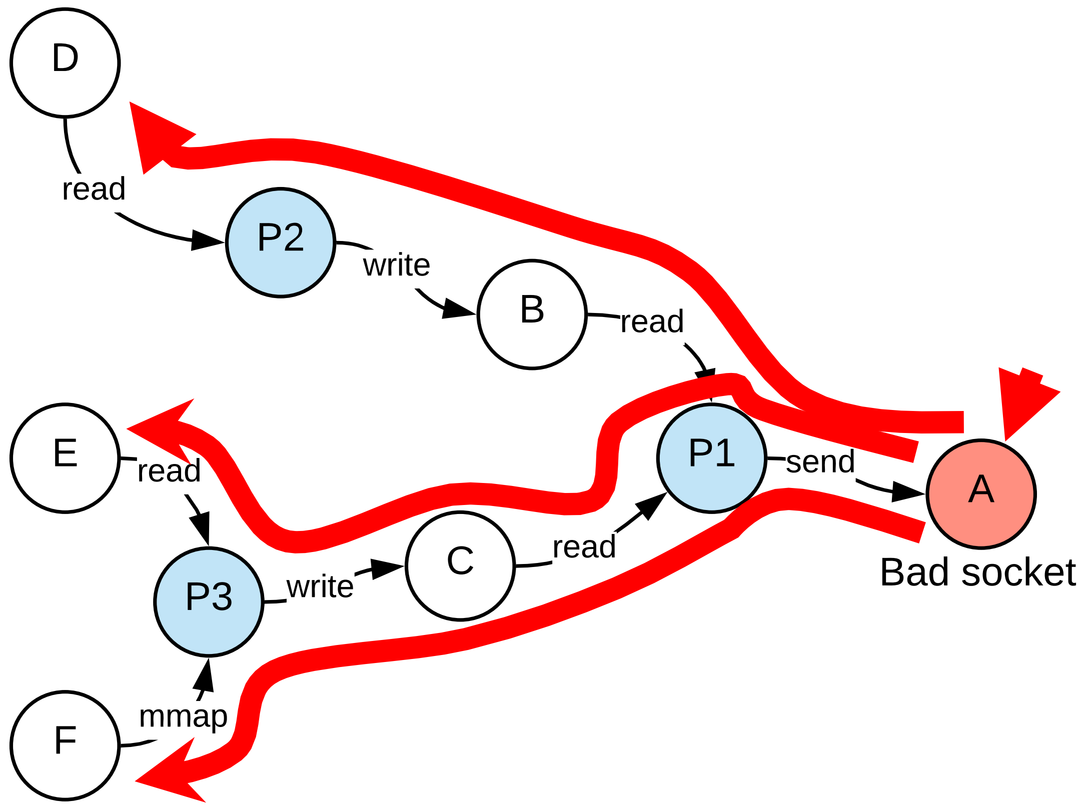
- Want to limit the size of the graph to the most interesting nodes
- Three criterion for starting the analysis
  - *External signals*: tips from other sources, CVEs, responsible disclosures, etc.
  - *Security policy*: violations to a certain policy are interesting points for looking into
  - *Customized comparisons*: compare hashes of downloaded files

PRUNING II: Reachability Analysis

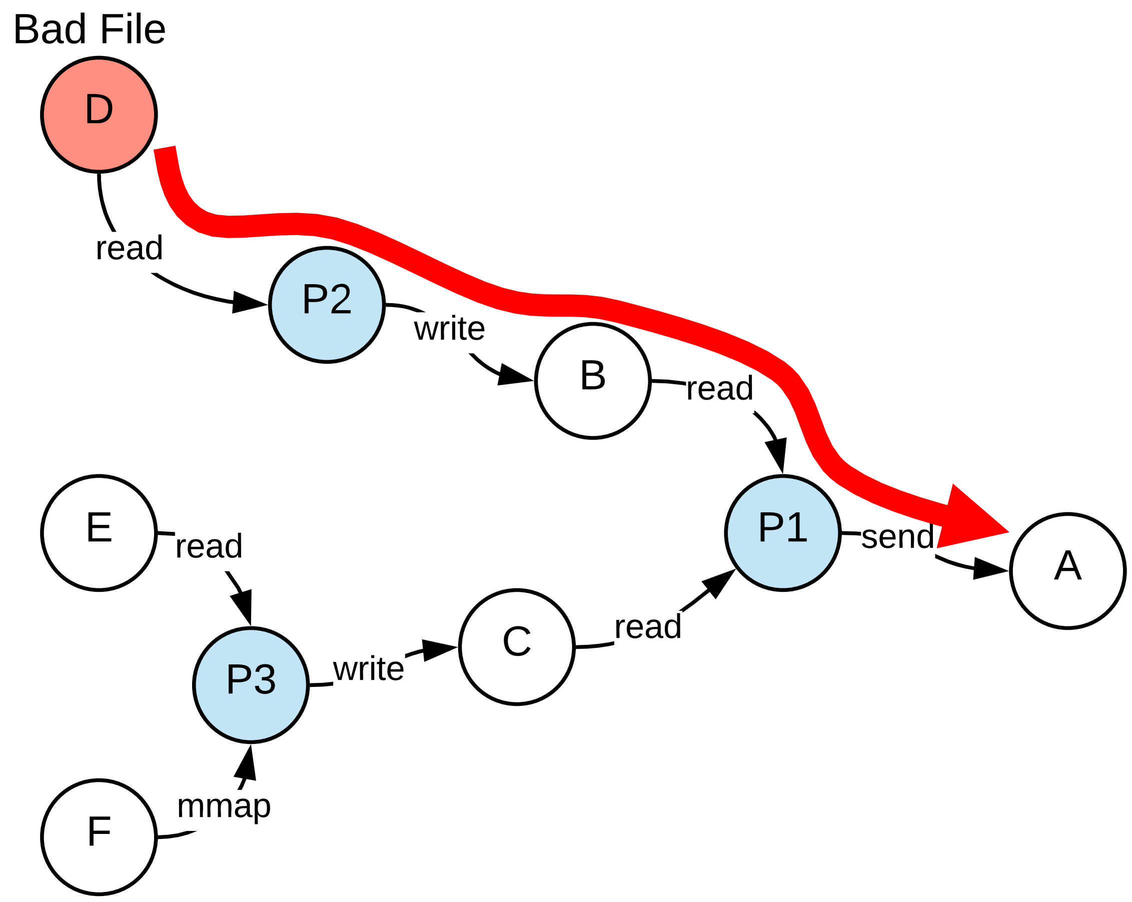
- Starting from trigger points (points of interest)
  - Determine the next set of interesting poinst

- Forward reachability
- Backward reachability
- Point-to-point: Forward & Backward
- Heuristic *interference* analysis

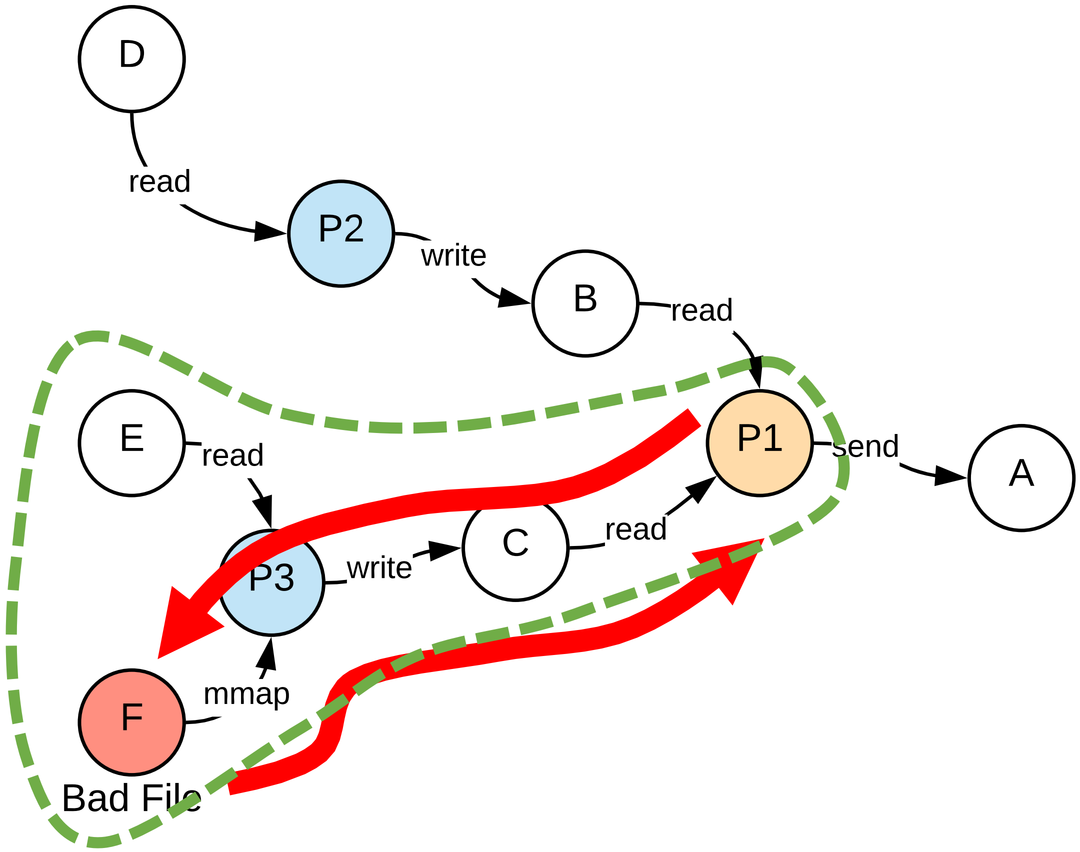
Backward Reachability Analysis



Forward Reachability Analysis

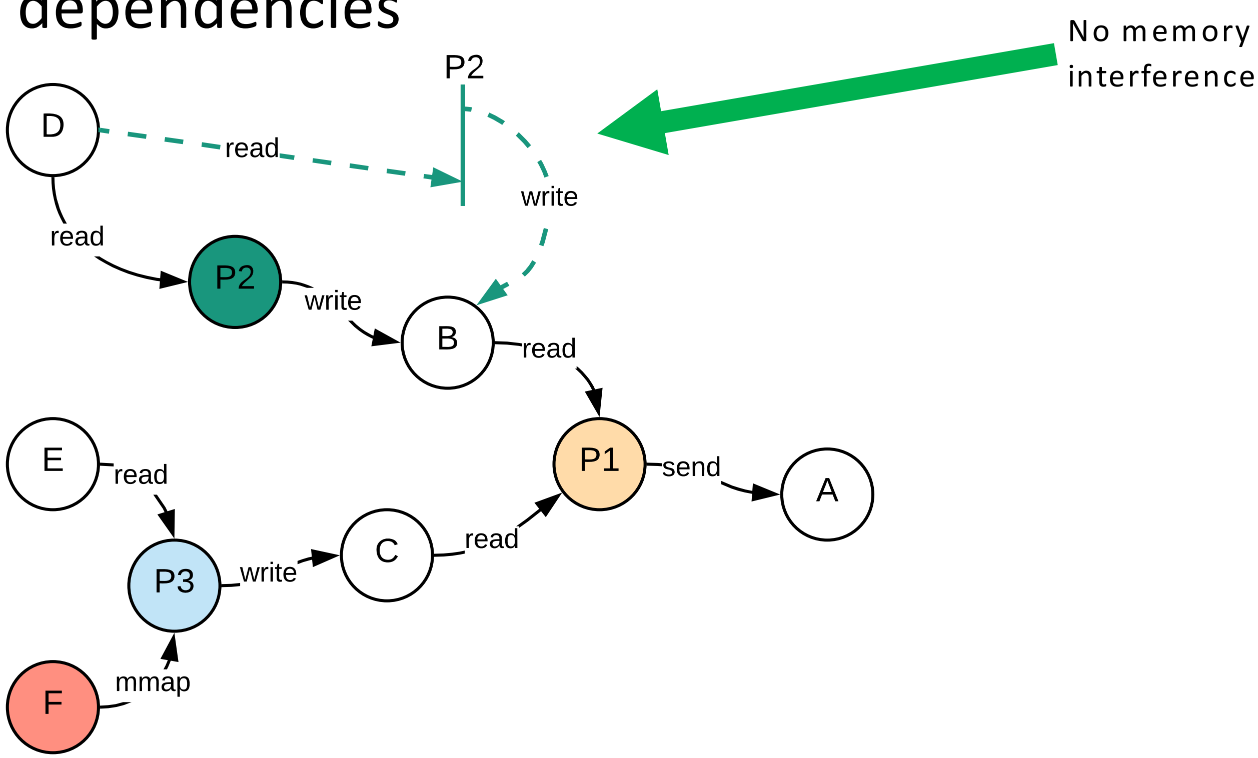


P2P Reachability



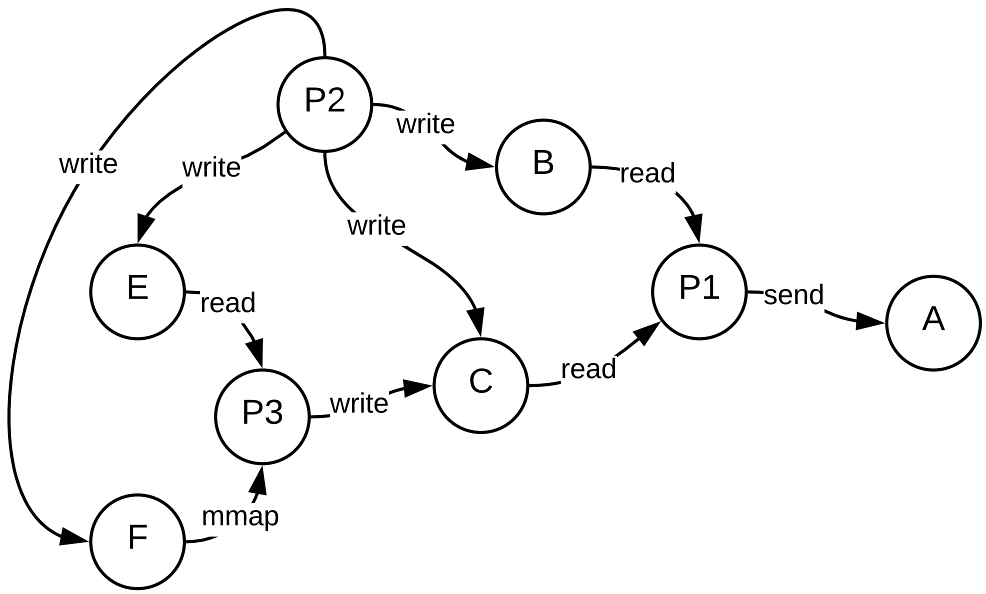
Interference Pruning

- Track *read-after-writes* using syscall timestamps
  - Remove false dependencies



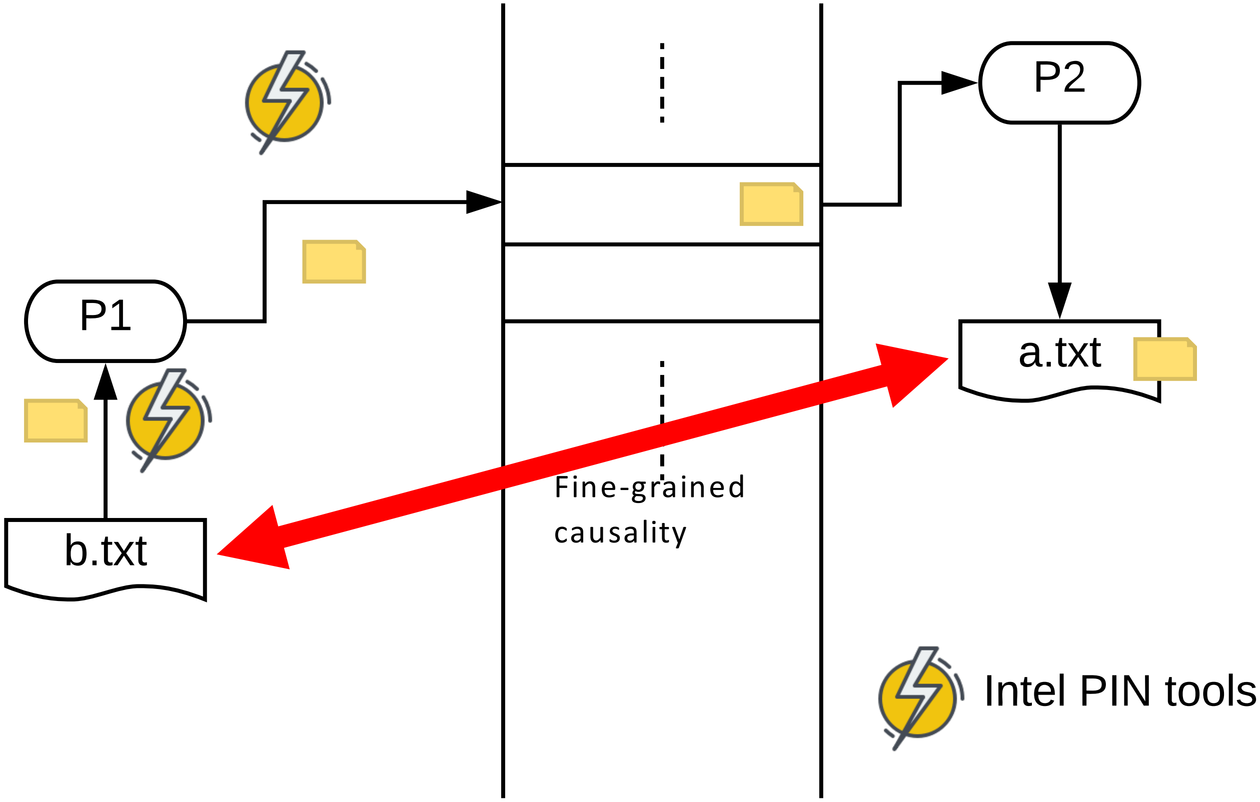
Digression

- High dependence on the structure of the graph
- What about loops?
- Processes that touch system files
  - `/etc, /var, /sys, …`



Taint Analysis Primer

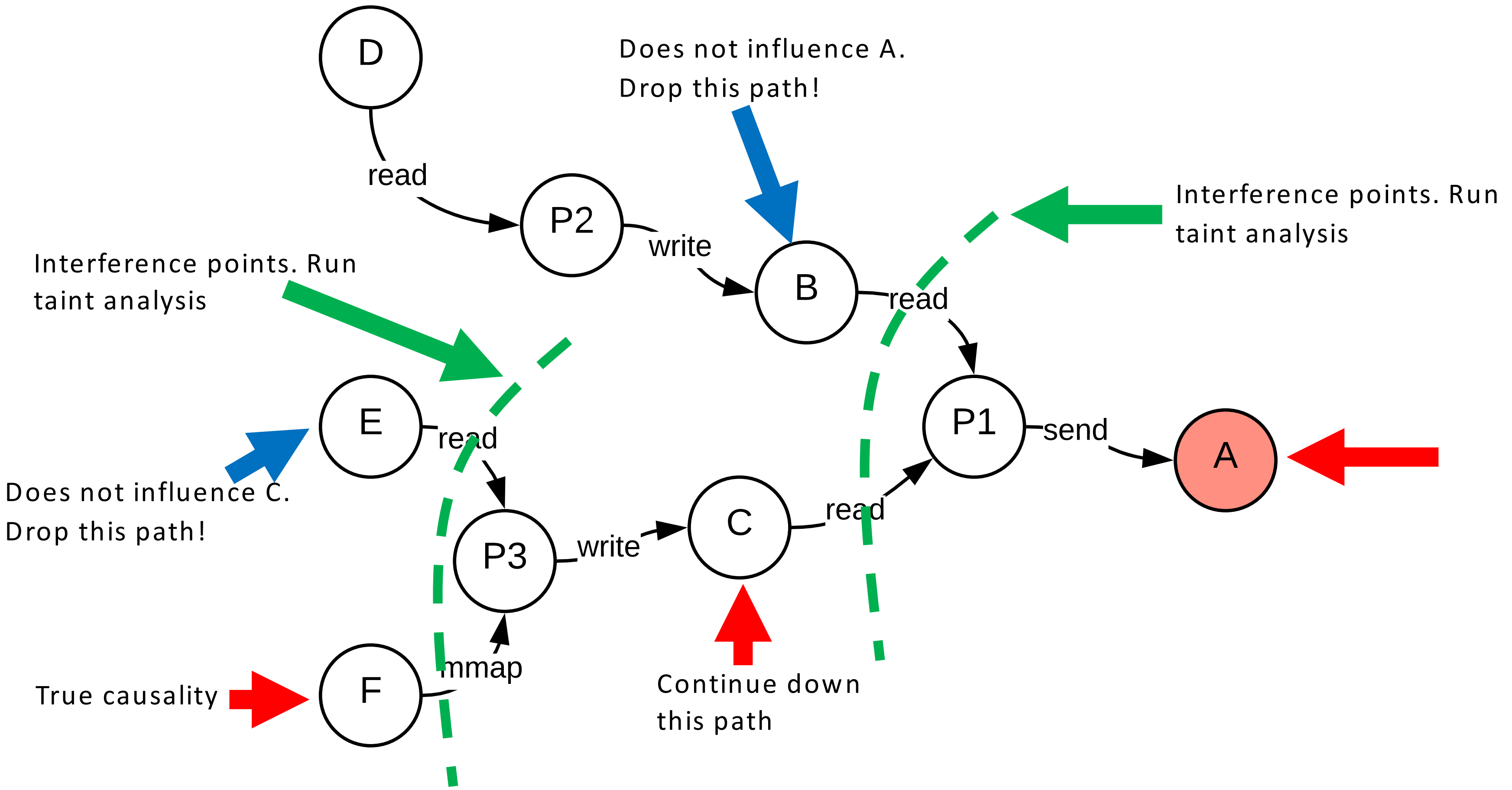
- A process level PET scan



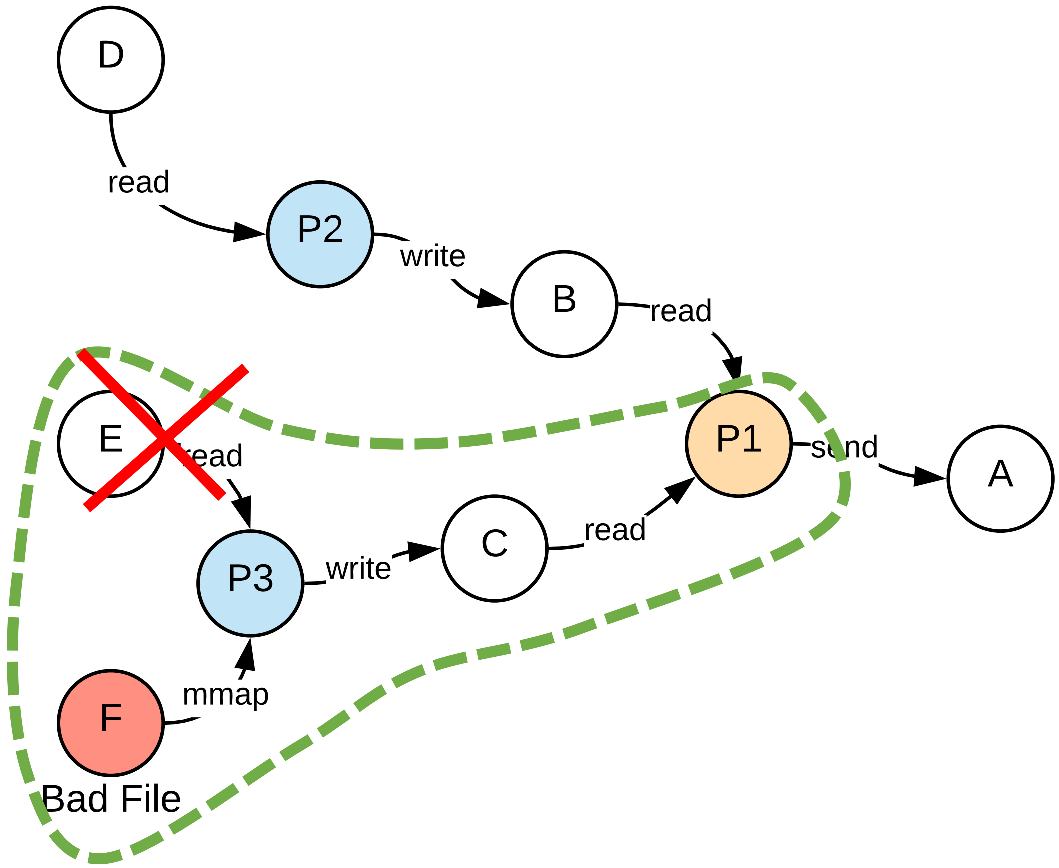
Selective DIFT

- Use the outcomes of the reachability analysis and trigger points
  - Start from interference points
- Refinement for
  - downstream causality,
  - upstream causality,
  - and point to point causality
- Run taint analysis for different processes independently
  - Cache results for improved performance

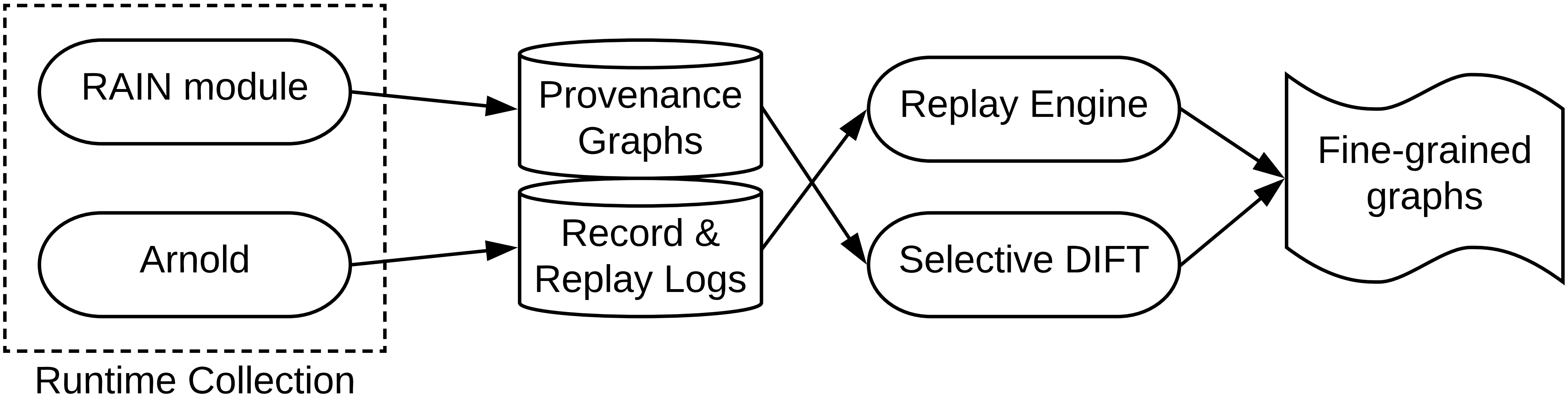
DIFT: Upstream Refinement



P2P Refinement



Story Recap


RAIN module
Arnold
Runtime Collection
Provenance Graphs
Record & Replay Logs
Replay Engine
Selective DIFT
Fine-grained graphs

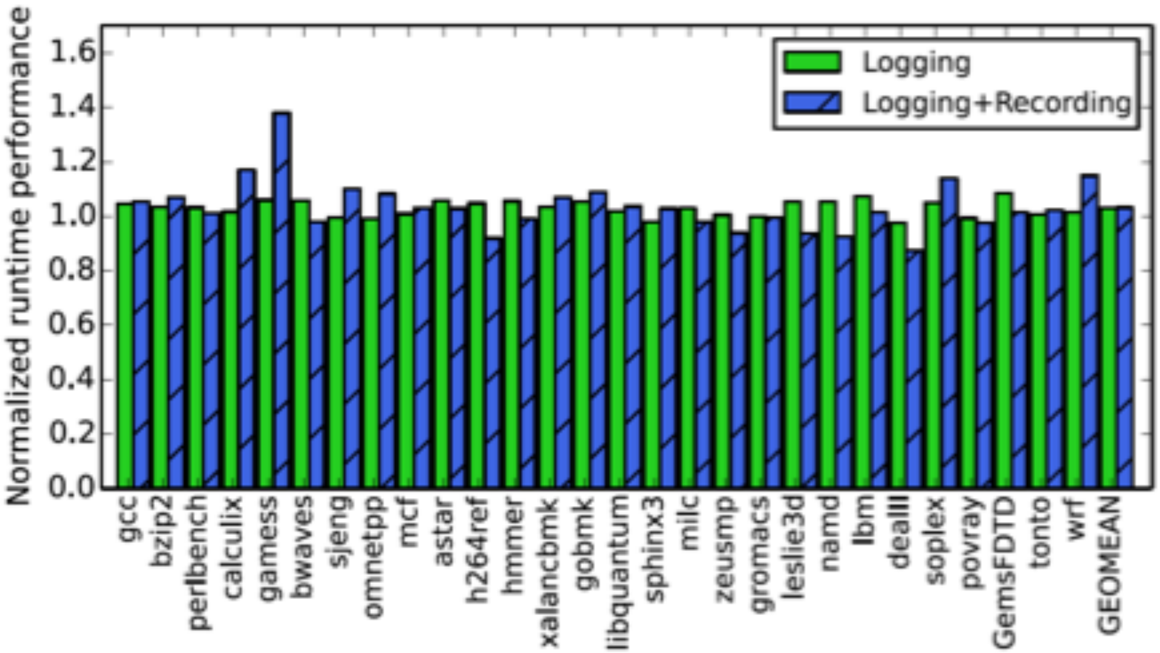
Results: Accuracy

Analysis Stages		Coarse Level Pruning			Fine Level Refinement					False Positive Rate		
AudioGrab	A(P-Dn)	4,909/33,382	415/3,394	8.5%/10.1%	31/161	7.4%/4.7%	-	-	-	**48.2%**	**0.0%**	**100.0%**
(30m)	A(P-P)		230/1,392	4.7%/4.2%	84/519	36.5%/37.3%	22	18	81.8%	**29.3%**	**0.0%**	**100.0%**

"In addition, the point-to-point analysis between the "NetRecon.log" and neighboring hosts shows the effectiveness of RAIN involving control flow dependency"

"When we took a closer look at the DIFT, we observed that "over-tainting" situation that occurs during control flow-based propagation which is a know limitation of DIFT".

Results: Performance Hit



	Storage usage (MB)		
Case	**Log**	**Record**	**Total**
MotivExp	45.6	155	201
NetRecon	29	137.1	166
ScreenGrab	16.6	97.3	114
CameraGrab	15.8	89.2	105
AudioGrab	18.2	115.4	133
Libc compiliation	327	413	740

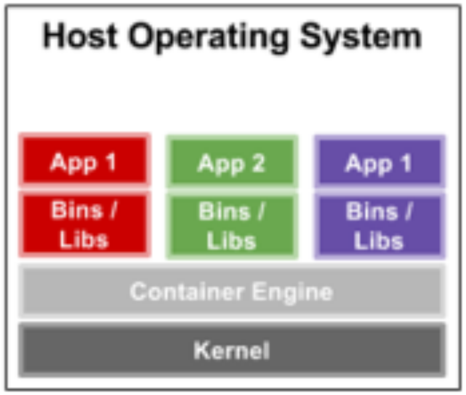
Limitations

- Storage overhead
- Over-tainting issue due to control flow dependencies
- Kernel is a point of trust
  - What if exploit is in libc but logging is intact?

Questions

- Attack that exploits a certain race condition?
  - Arnold is having an affair:

    "In the presence of data races, the replayed execution may diverge from the recorded one"[1]
- Does record and replay as described work with containers?



Containerized Applications

[1] Devecsery, David, et al. "Eidetic Systems." *OSDI*. Vol. 14. 2014.