Progressive Processing of System-Behavioral Query

12/12/2019

Jiaping Gui[*], Xusheng Xiao[‡], Ding Li[*], Chung Hwan Kim[*], and Haifeng Chen[*]

[*]NEC Laboratories America, Inc.

[‡]Case Western Reserve University

NEC Laboratories America

Relentless passion for innovation

Motivation

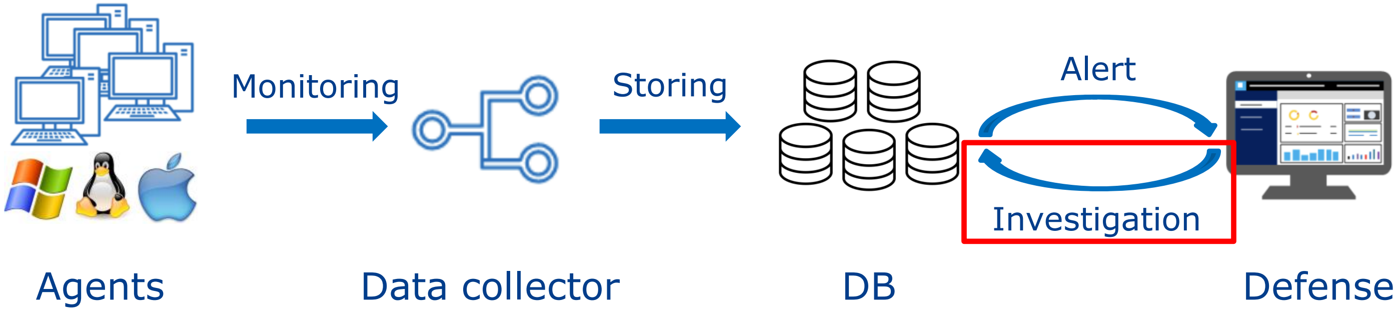
- Threat detection and investigation is an important security solution in enterprises

cisco Security

McAfee SECURE™

Symantec

splunk>



Monitoring

Storing

Alert

Investigation

Agents

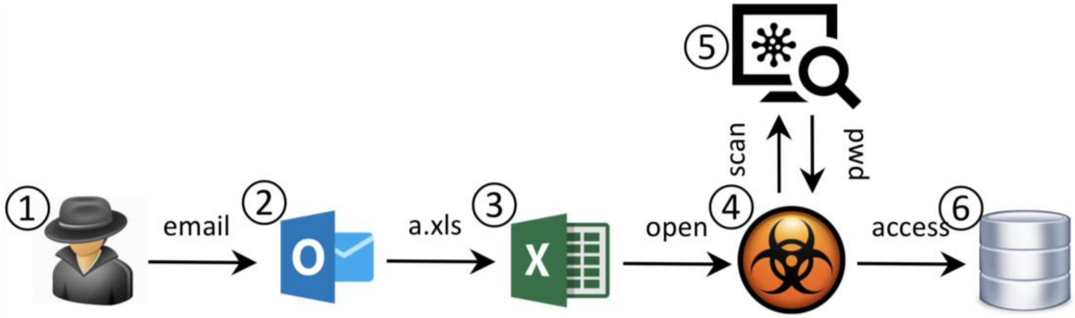
Data collector

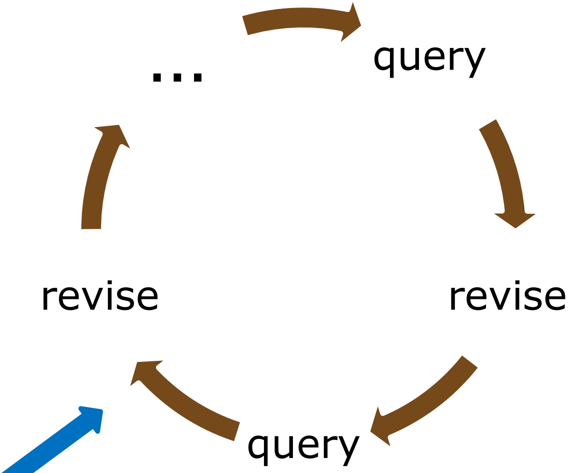
DB

Defense

Motivation

- Alert investigation





...

query

revise

revise

query

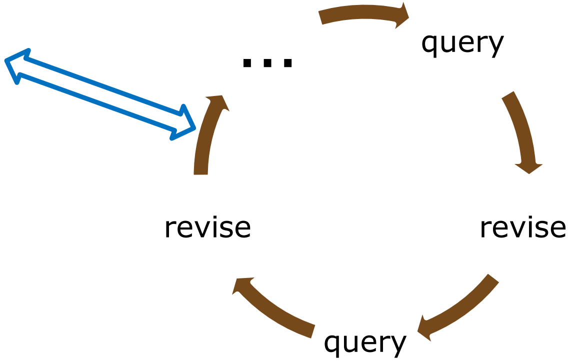
- Process
  - Query 1: select processes that accessed sensitive data in DB
  - Query 2: check whether unsigned program executed probing commands
  - Query 3: get source process that opened/created unsigned program
  - …

May take a long execution time

Challenges

- Long waiting time for even a single query
  - A huge amount of data in DB
    - > 100GB/200 computers/day
  - Query multiple hosts' or multiple days' data
    - Some advanced attack behaviors may span over several months
    - Check other machines if the same suspicious behaviors exist

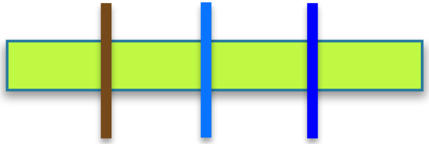
- Making interactive querying difficult



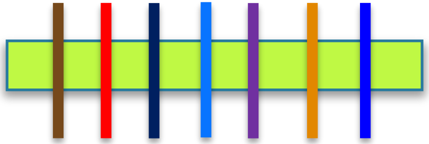
query
revise
query
revise
...

Challenges

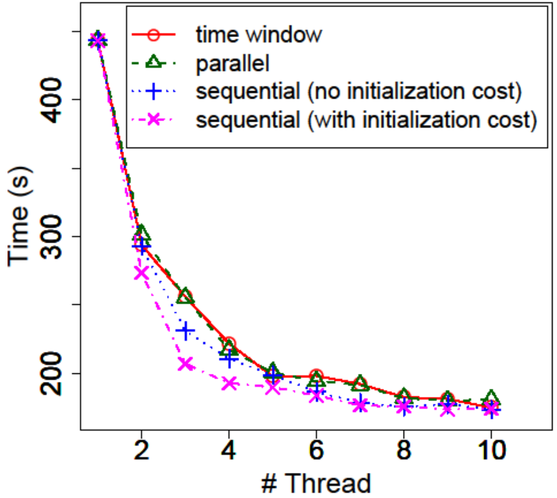
- Optimize the query execution
  - ○ > 30% improvement (parallel execution)



1-host query into 4 sub-queries



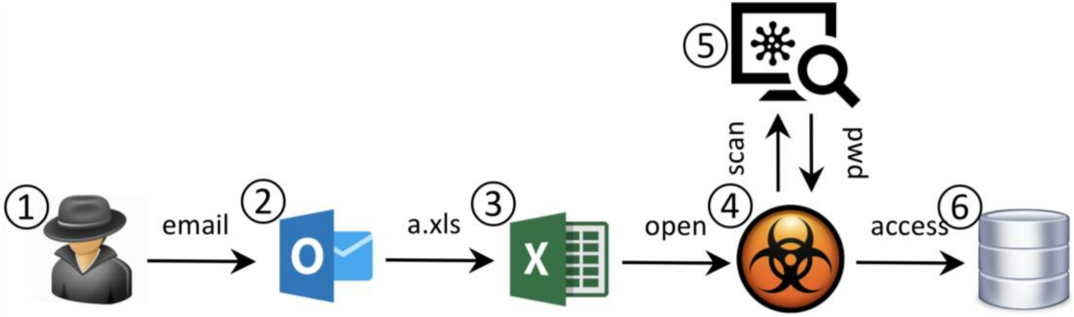
1-host query into 8 sub-queries



- Some sub-queries may still take a long time even with optimization
  - ○ Especially when querying multiple hosts’/days’ data
  - ○ Bounded by hardware (bottleneck)
    - ❖ **Sub-query costs:** DB connection, query parsing, thread overhead
    - ❖ **Hardware limitation:** CPU, disk, etc.

Insight

- Partial results are very helpful to make a decision!

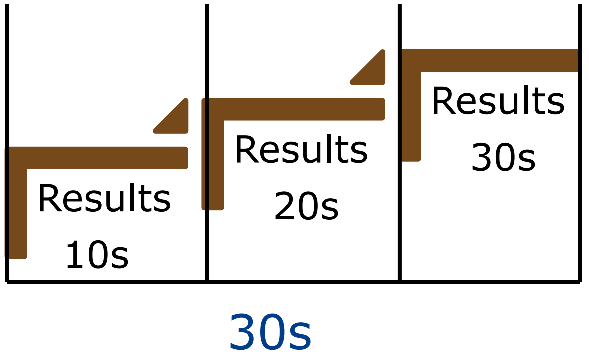


- Process
  - Query 1: select processes that accessed sensitive data in DB
  - Query 2: check whether unsigned program executed probing commands
  - Query 3: get source process that opened/created unsigned program …

Pause and revise query when seeing unsigned program

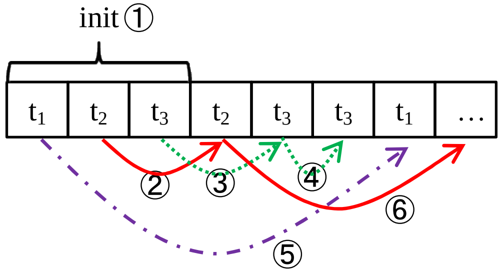
Approach

- Progressive Querying
  - Progressively update results during the execution instead of until the end



❖ Quality metrics

- ○ Q.1: results updated within the update cycle
- ○ Q.2: small overhead on the total execution time



Progressive Querying: straightforward solutions

- Naïve solution
  - Partition the query into sub-queries, each with time window 1s
    - e.g., 1-day query = 3600*24 subqueries
  - >28hrs (1 worker thread)
  - 6.7hrs (5 worker threads)
  - ➢ **Q.1: update fast**
  - ➢ ***Q.2: unacceptable overhead***

- Whole-query update
  - # sub-queries = # worker threads
  - 532s (1 worker thread)
  - 214s (5 worker threads)
  - ➢ ***Q.1: only 1 update***
  - ➢ **Q.2: low overhead**

More intelligent solutions are desired!

- Ideal: sub-queries finish exactly before each update cycle
- Practical: average finish time is close to update cycle

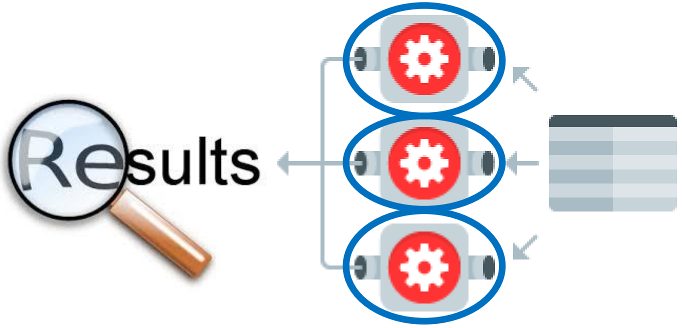
Progressive Querying

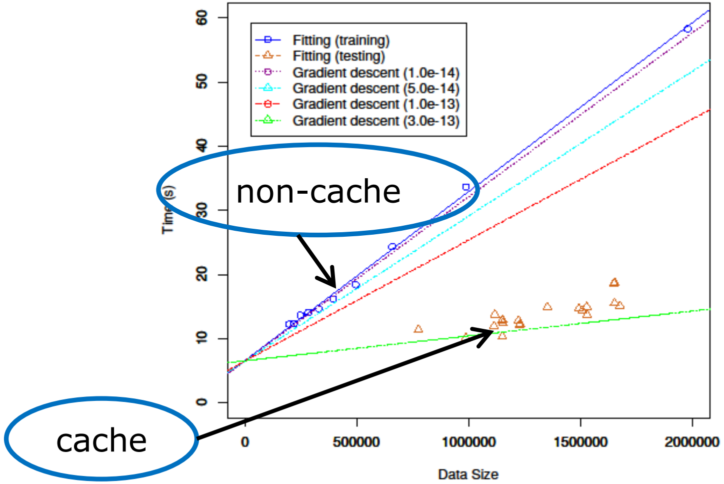
- Intelligent solutions
  - Query partition
    - Fixed workload
    - Fixed time window
    - Adaptive learning

❖ Fixed Strategy: cache mechanism / system dynamics are not considered

- ○ Event processing rate (#events/s): cache >> non cache
- ○ Sub-queries’ execution time varies much → average time is far from update frequency

Sub-queries





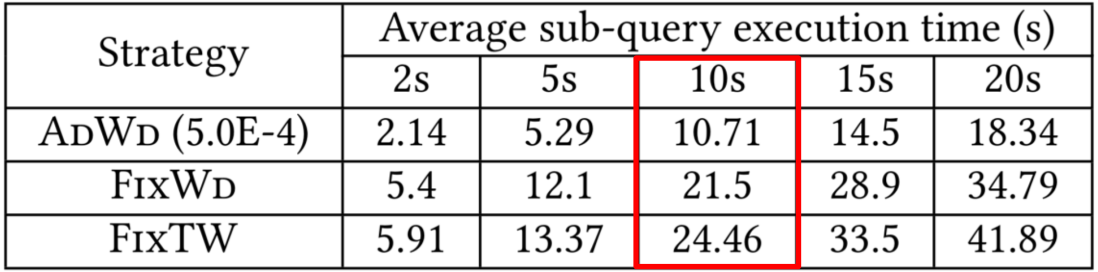
Progressive Querying

- Adaptive learning → spatial & temporal
  - Goal: adjust event processing rate dynamically
    - Cache
    - Non-cache
  - Gradient descent algorithm
    - Learn different event processing rates

➢ **Reflect the system runtime environment**

Results: Progressive Querying

- Comparison
  - Fixed time window
  - Fixed workload
  - Adaptive learning



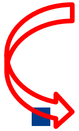
Strategy	Average sub-query execution time (s)				
	2s	5s	10s	15s	20s
AdWd (5.0E-4)	2.14	5.29	10.71	14.5	18.34
FixWd	5.4	12.1	21.5	28.9	34.79
FixTW	5.91	13.37	24.46	33.5	41.89

Average sub-query execution time

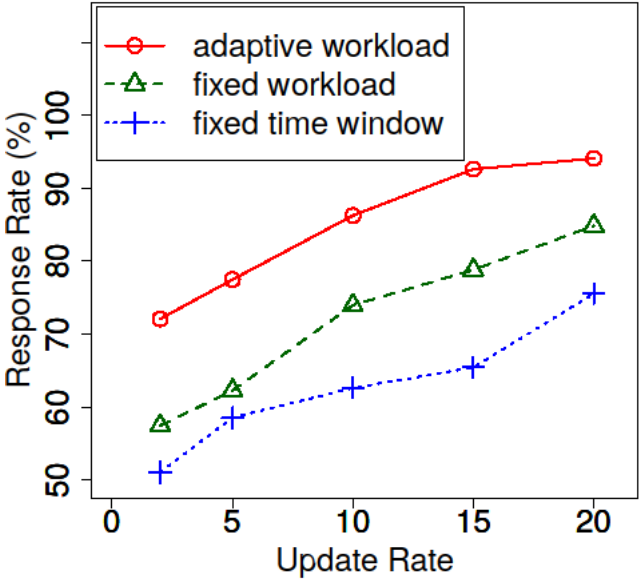
- Adaptive learning
  - **Closest** proximity of average sub-query time to update frequency
    - E.g., with update cycle 10s, if we have 1000 sub-queries to execute, it can save us > 3 hours compared to fixed strategy

Results: Progressive Querying

- Comparison
  - Fixed time window
  - Fixed workload
  - Adaptive learning



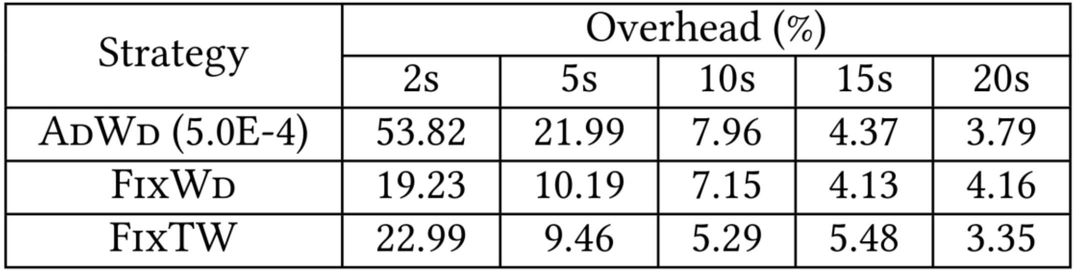
- Adaptive learning
  - **Closest** proximity of average sub-query time to update frequency
  - **Best** response rate: result update at each cycle



Response rate

Results: Progressive Querying

- Comparison
  - Fixed time window
  - Fixed workload
  - Adaptive learning



Strategy	Overhead (%)				
	2s	5s	10s	15s	20s
AdWd (5.0E-4)	53.82	21.99	7.96	4.37	3.79
FixWd	19.23	10.19	7.15	4.13	4.16
FixTW	22.99	9.46	5.29	5.48	3.35

Overhead

- Adaptive learning
  - **Closest** proximity of average sub-query time to update frequency
  - **Best** response rate: result update at each cycle
  - **Comparable** overhead

Conclusion

- A systematic approach to optimize query execution on suspicious system behaviors
  - Parallel execution
  - Performance: sequential with cost >= Sequential >= Parallel >= Time window
- A comprehensive comparison on progressively processing return results
  - Fixed time window (processing rate & data rate)
  - Fixed workload (all hosts/single host)
  - Adaptive (different learning rates) → best performance

Q&A

\Orchestrating a brighter world

www.nec-labs.com