SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection

Peng Gao[1], Xusheng Xiao[2], Ding Li[3], Zhichun Li[3], Kangkook Jee[3],
Zhenyu Wu[3], Chung Hwan Kim[3], Sanjeev R. Kulkarni[1], Prateek Mittal[1]

[1]Princeton University

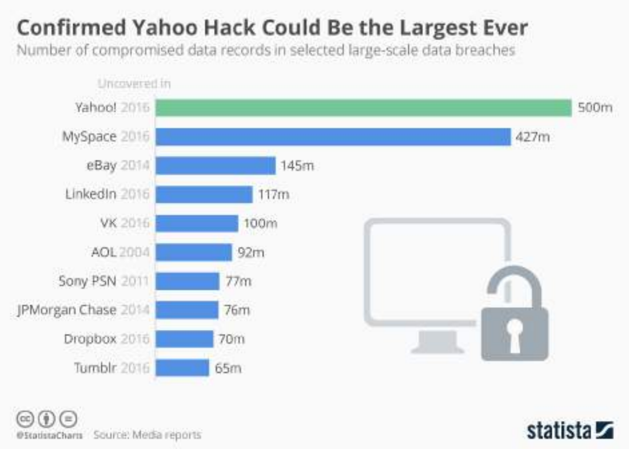
[2]Case Western Reserve University

[3]NEC Laboratories America, Inc.

The Equifax Data Breach

CYBERATTACK
PERSONAL INFORMATION EXPOSED
EQUIFAX
AS MANY AS
143
MILLION
CUSTOMERS

Impact of Advanced Persistent Threat (APT) Attack

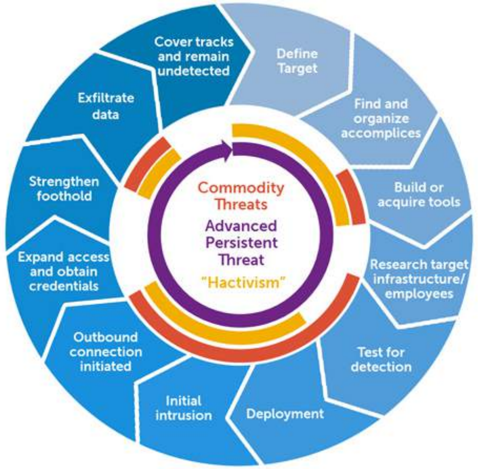

Confirmed Yahoo Hack Could Be the Largest Ever

Number of compromised data records in selected large-scale data breaches

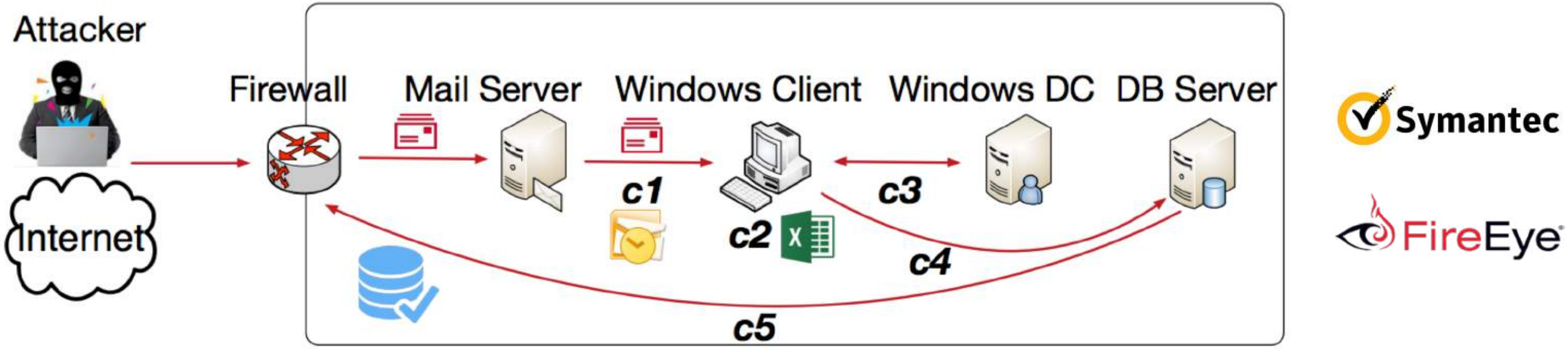
@StatistaCharts Source: Media reports

statista

- Advanced: sophisticated techniques, e.g., exploiting multiple vulnerabilities
- Persistent: adversaries are continuously monitoring and stealing data from the target
- Threat: strong economical or political motives

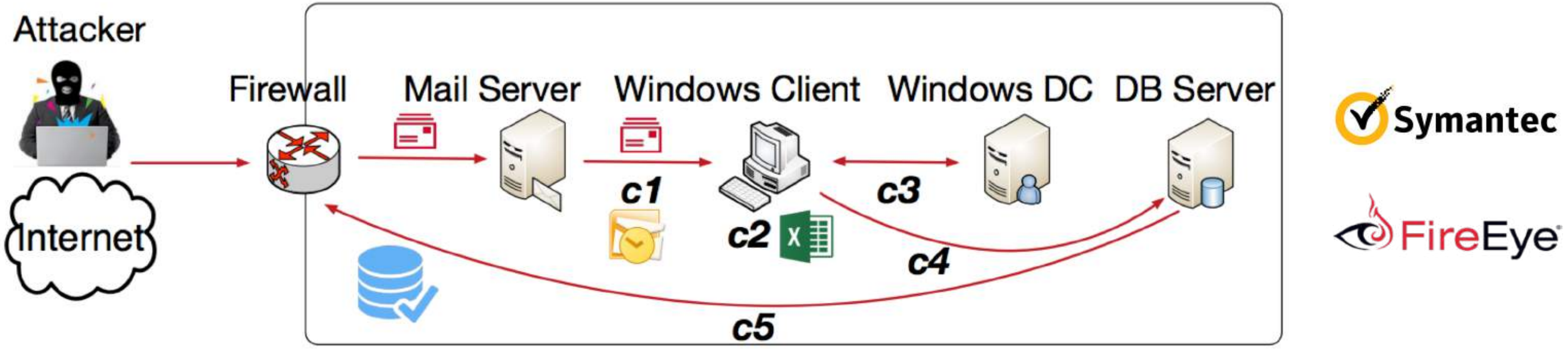

Define Target
Find and organize accomplices
Build or acquire tools
Research target infrastructure/ employees
Test for detection
Deployment
Initial intrusion
Outbound connection initiated
Expand access and obtain credentials
Strengthen foothold
Exfiltrate data
Cover tracks and remain undetected
Commodity Threats
Advanced Persistent Threat
"Hactivism"

APT Attack: Case Study



- **c1 Initial Compromise**: Attacker sends a crafted e-mail to the victim, which contains an Excel file with a malicious macro embedded
- **c2 Malware Infection**: Victim opens the file and runs the macro, which downloads and executes a malware to open a backdoor
- **c3 Privilege Escalation**: Attacker enters the victim's machine through the backdoor and runs the database cracking tool to obtain database credentials
- **c4 Penetration into Database Server**: Attacker penetrates into the database server and drops another malware to open another backdoor
- **c5 Data Exfiltration**: Attacker dumps the database content and sends the dump back to his host

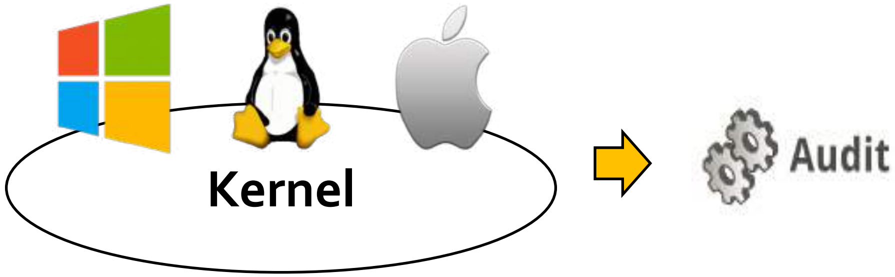
APT Attack: Case Study



- Multiple steps exploiting different types of vulnerabilities in the system, exhibiting different abnormal behaviors
  - Known malicious behaviors, e.g., “cmd.exe” starts “gsecdump.exe” (**c3**)
  - Abnormal data transfers, e.g., “sqlservr.exe” transfers large data to external IP, causing large network spikes (**c5**)
  - Abnormal process creations, e.g., “excel.exe” starts “java.exe” (**c2**)

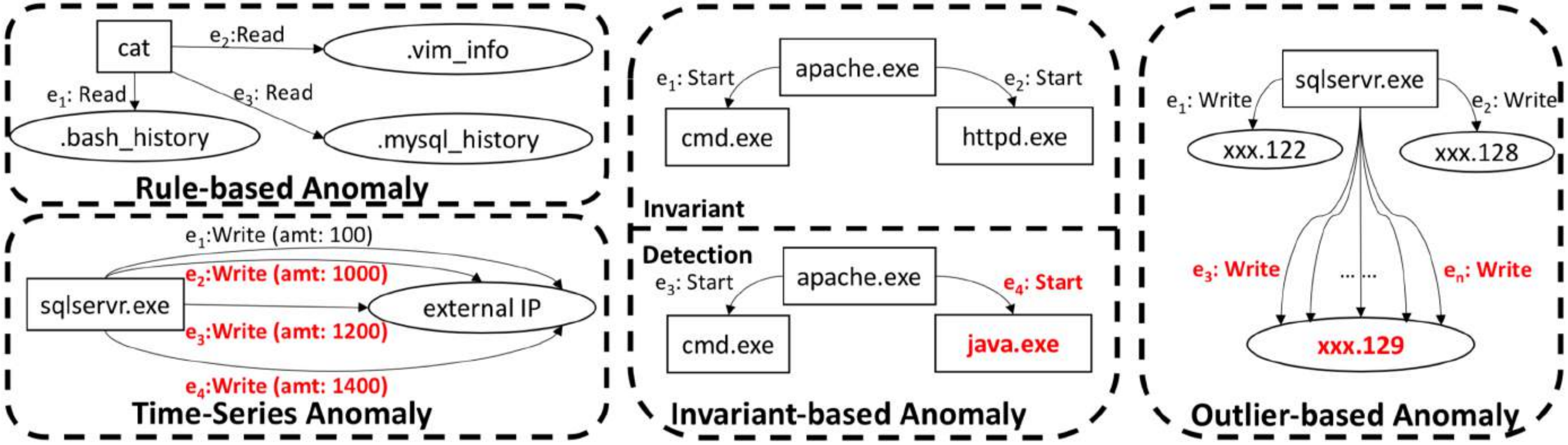
Ubiquitous System Monitoring

- Recording system behaviors from kernel
  - Unified structure of logs: not bound to applications



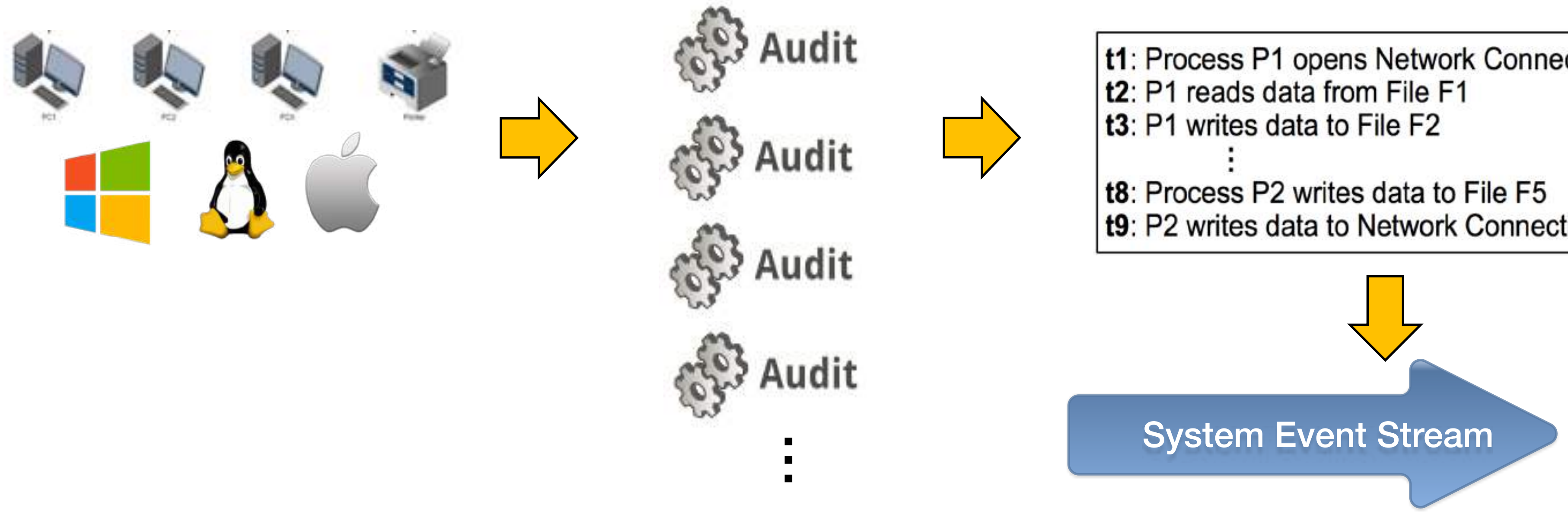
- System activities w.r.t. system resources
  - System resources (system entities): processes, files, network connections
  - System activities (system events): file events, process events, network events
    - Format: <subject, operation, object>, e.g., proc p1 read file f1
- Enabling timely anomaly detection via querying the real-time stream of system monitoring data
  - Continuous queries

Challenge 1: Attack Behavior Specification



- **Rule-based anomaly**: behavioral rules of system activities and their relationships
- **Time-Series anomaly**: states definition and history states comparison
- **Invariant-based anomaly**: invariant definition, training, and violation checking
- **Outlier-based anomaly**: peer states comparison

Challenge 2: Timely “Big Data” Security Analysis



Audit

Audit

Audit

Audit

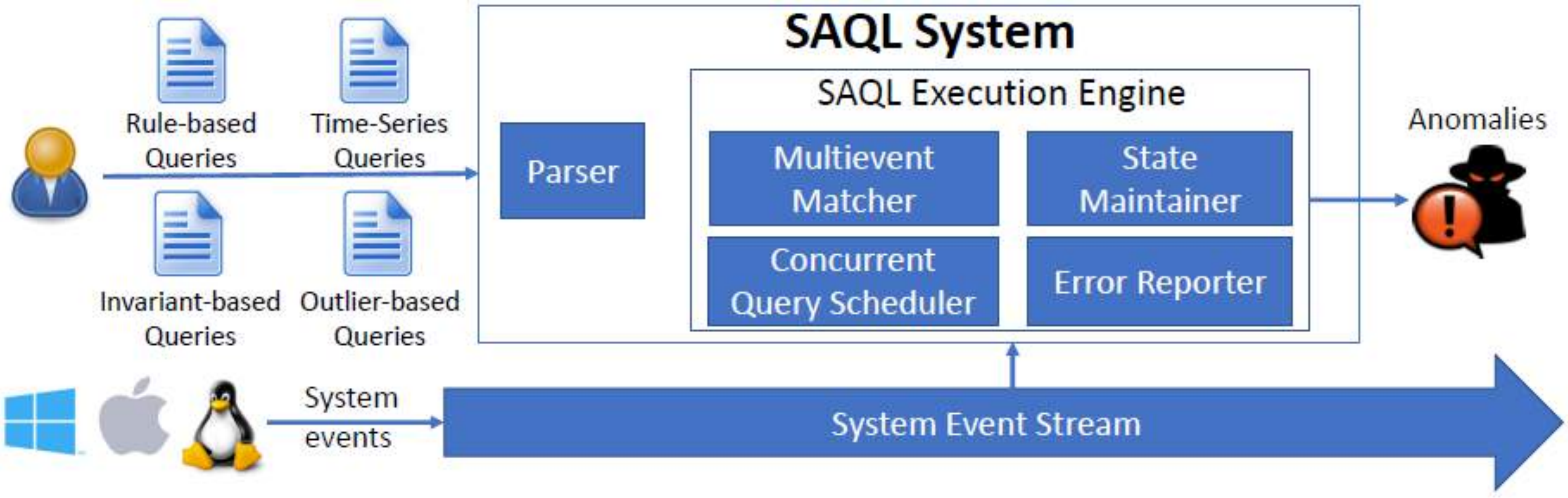
⋮

t1: Process P1 opens Network Connection N1
t2: P1 reads data from File F1
t3: P1 writes data to File F2
⋮
t8: Process P2 writes data to File F5
t9: P2 writes data to Network Connection N2

System Event Stream

- System monitoring produces huge amount of system logs per day
  - ~50 GB for 100 hosts per day; throughput ~2500 system events/s (in typical computer science research lab environment)
- Executing multiple concurrent queries incurs considerable overhead

SAQL System

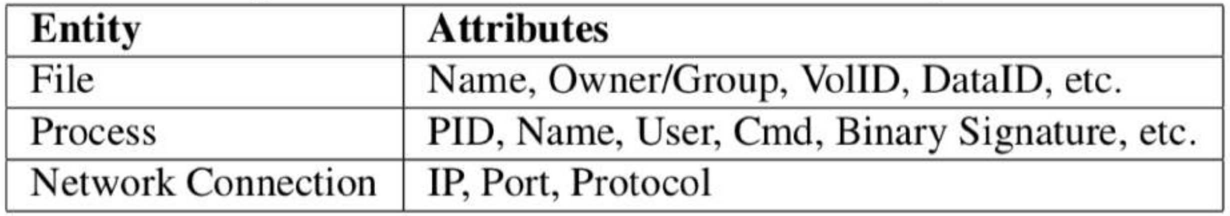


- Novel stream query system for abnormal system behavior detection
  - Build on top of existing mature tools (~50,000 lines of Java code)
    - System-level monitoring tools: auditd, ETW, Dtrace
    - Event stream management: Siddhi

Data Collection

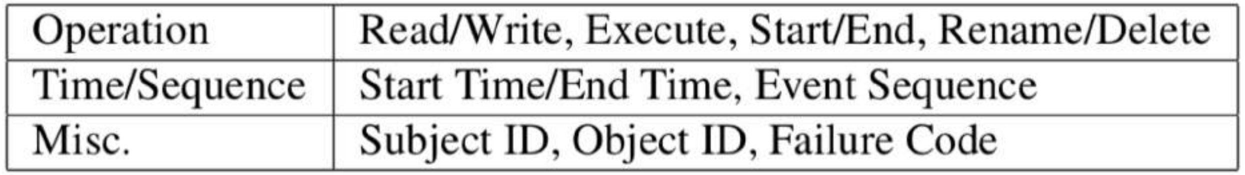
- Data collection agent: system calls as a sequence of system events
  - ➢ Windows: Event Tracing for Windows (ETW)
  - ➢ Linux: Audit Framework (auditd)
  - ➢ Mac: DTrace
- Collect critical attributes for security analysis

Table 1: Representative attributes of system entities



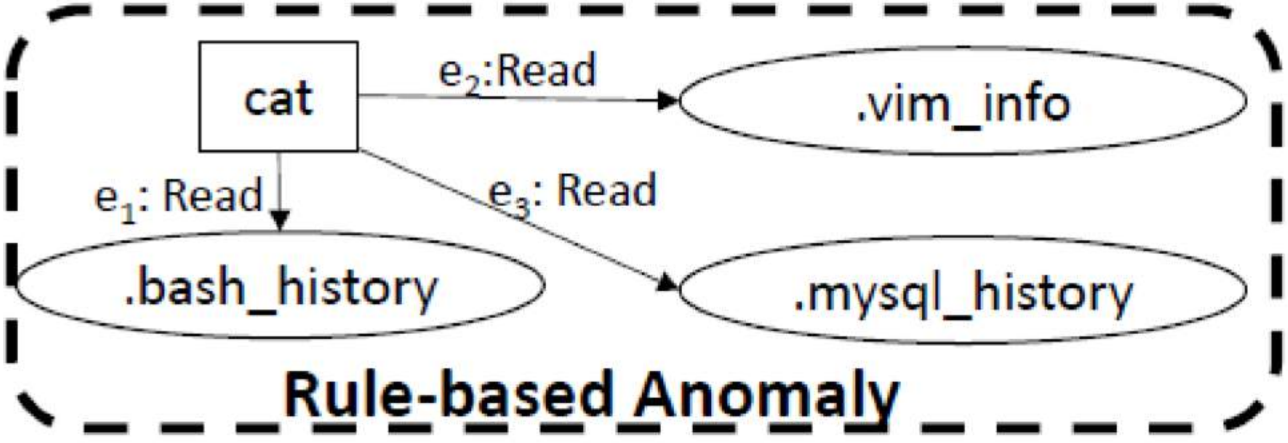
Entity	**Attributes**
File	Name, Owner/Group, VolID, DataID, etc.
Process	PID, Name, User, Cmd, Binary Signature, etc.
Network Connection	IP, Port, Protocol

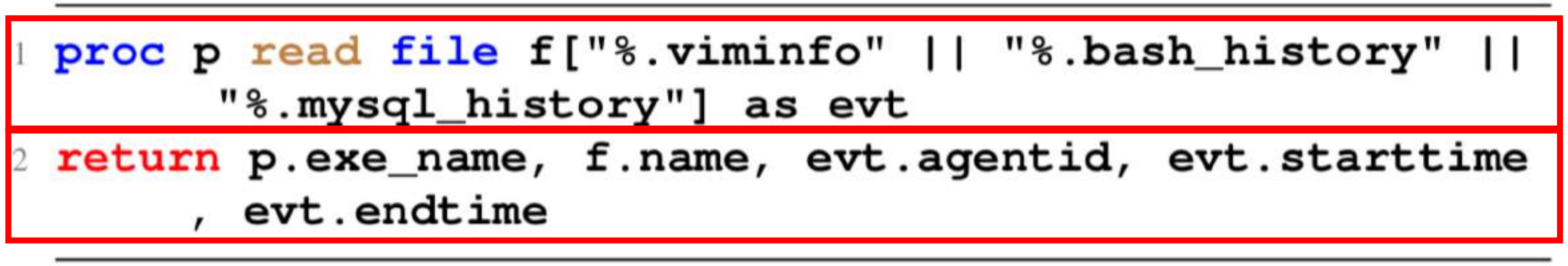
Table 2: Representative attributes of system events



Operation	Read/Write, Execute, Start/End, Rename/Delete
Time/Sequence	Start Time/End Time, Event Sequence
Misc.	Subject ID, Object ID, Failure Code

Rule-based Anomaly: Single-Event

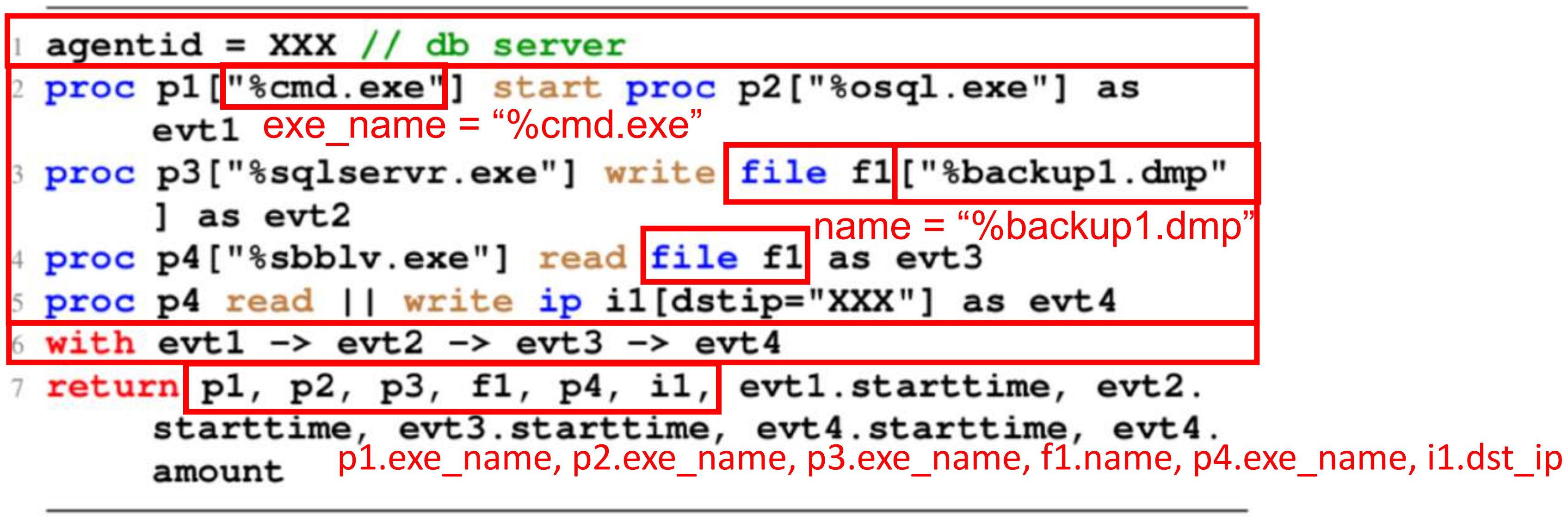




```
proc p read file f["%.viminfo" || "%.bash_history" ||
        "%.mysql_history"] as evt
return p.exe_name, f.name, evt.agentid, evt.starttime
        , evt.endtime
```

- Event pattern: <subject, operation, object>, attribute constraints, event ID
- Return attributes

Rule-based Anomaly: Multievent



```
agentid = XXX // db server
proc p1["%cmd.exe"] start proc p2["%osql.exe"] as
     evt1
proc p3["%sqlservr.exe"] write file f1["%backup1.dmp"
     ] as evt2
proc p4["%sbblv.exe"] read file f1 as evt3
proc p4 read || write ip i1[dstip="XXX"] as evt4
with evt1 -> evt2 -> evt3 -> evt4
return p1, p2, p3, f1, p4, i1, evt1.starttime, evt2.
     starttime, evt3.starttime, evt4.starttime, evt4.
     amount
```

exe_name = “%cmd.exe”

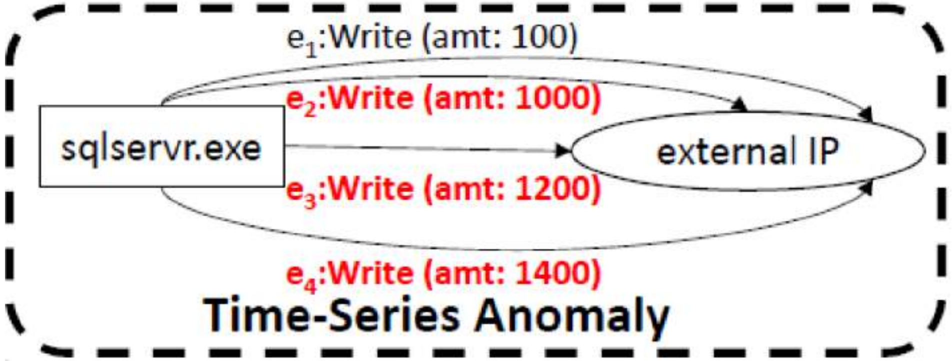
name = “%backup1.dmp”

p1.exe_name, p2.exe_name, p3.exe_name, f1.name, p4.exe_name, i1.dst_ip

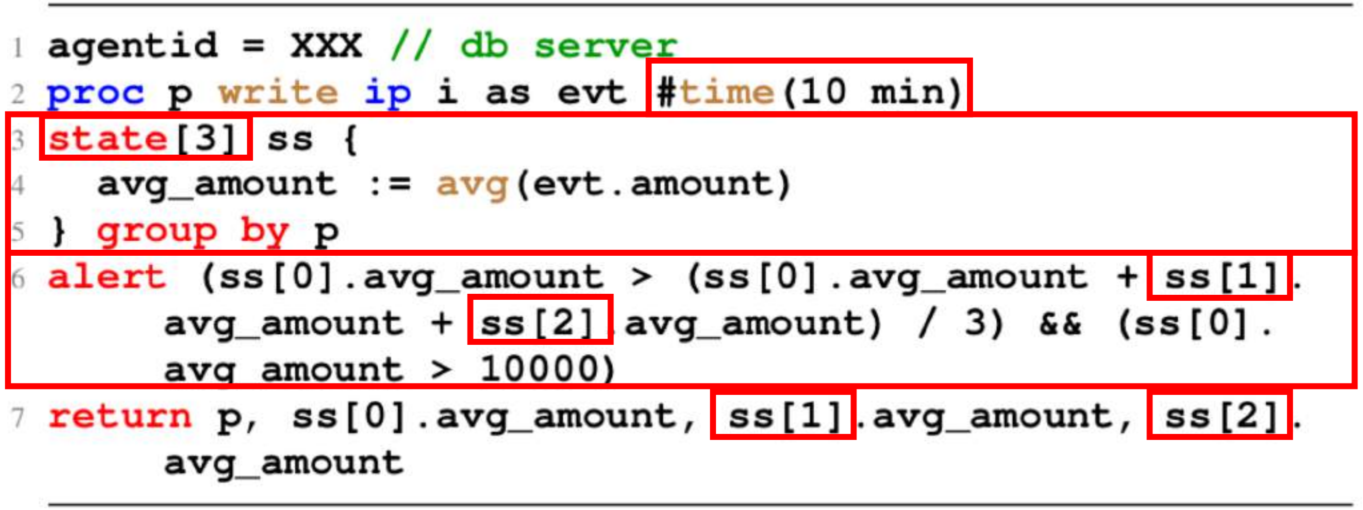
- Global constraints: e.g., agent ID
- Event patterns: <subject, operation, object>, attribute constraints, event ID
- Temporal relationships: enforce the event order
- Attribute relationships: e.g., two events linked by the same entity
- Syntax shortcuts: e.g., context-aware attribute inference

Time-Series Anomaly

- Sliding windows
- Aggregation states
- History states access
- Time-series anomaly models (e.g., SMA3)



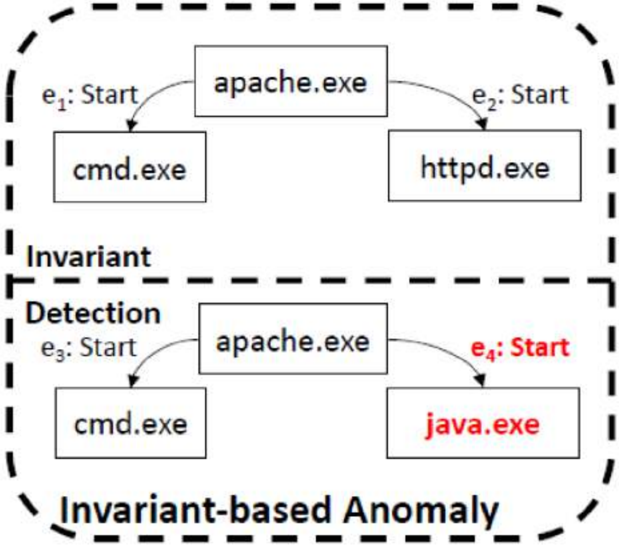
Existing systems lack the explicit support for stateful computation in sliding windows

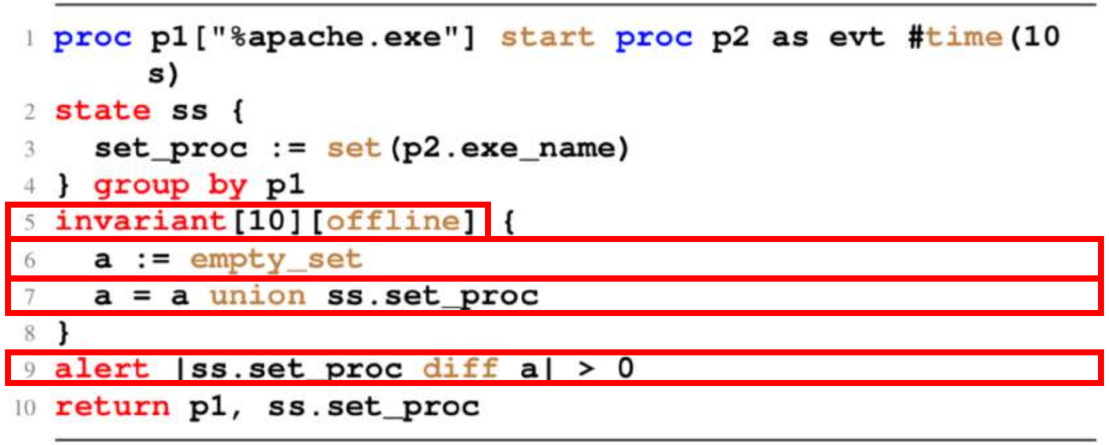


```
agentid = XXX // db server
proc p write ip i as evt #time(10 min)
state[3] ss {
  avg_amount := avg(evt.amount)
} group by p
alert (ss[0].avg_amount > (ss[0].avg_amount + ss[1].
      avg_amount + ss[2].avg_amount) / 3) && (ss[0].
      avg_amount > 10000)
return p, ss[0].avg_amount, ss[1].avg_amount, ss[2].
      avg_amount
```

Invariant-based Anomaly

- Invariants definition
- Invariants update
- Offline/online training
- Invariant-based anomaly models

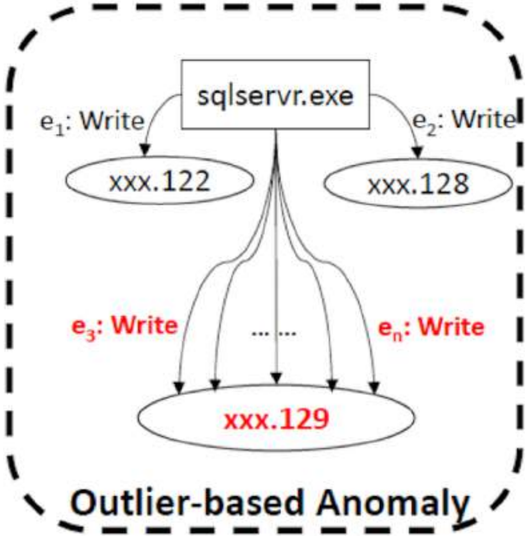


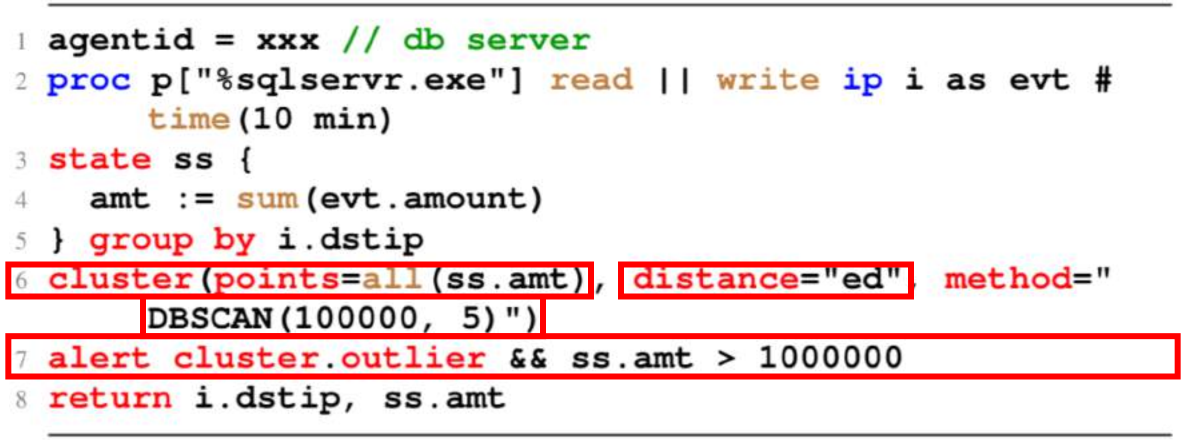


```
proc p1["%apache.exe"] start proc p2 as evt #time(10
     s)
state ss {
  set_proc := set(p2.exe_name)
} group by p1
invariant[10][offline] {
  a := empty_set
  a = a union ss.set_proc
}
alert |ss.set_proc diff a| > 0
return p1, ss.set_proc
```

Outlier-based Anomaly

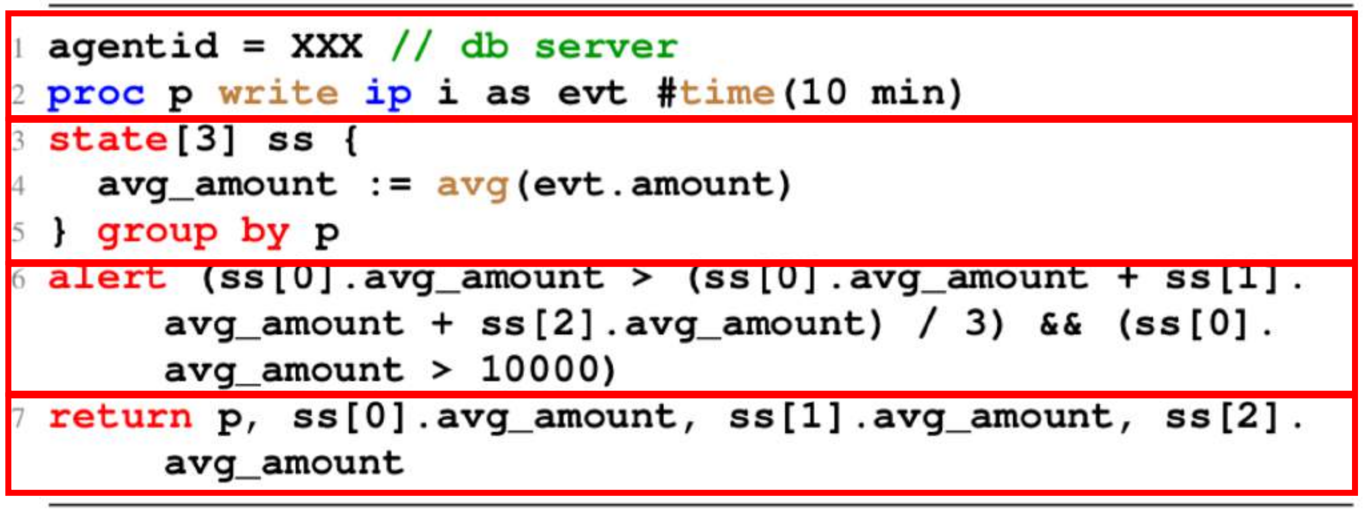
- Cluster definition
- Distance metric
- Clustering method
- Outlier-based anomaly models





```
agentid = xxx // db server
proc p["%sqlservr.exe"] read || write ip i as evt #
      time(10 min)
state ss {
  amt := sum(evt.amount)
} group by i.dstip
cluster(points=all(ss.amt), distance="ed", method="
      DBSCAN(100000, 5)")
alert cluster.outlier && ss.amt > 1000000
return i.dstip, ss.amt
```

SAQL Execution Engine

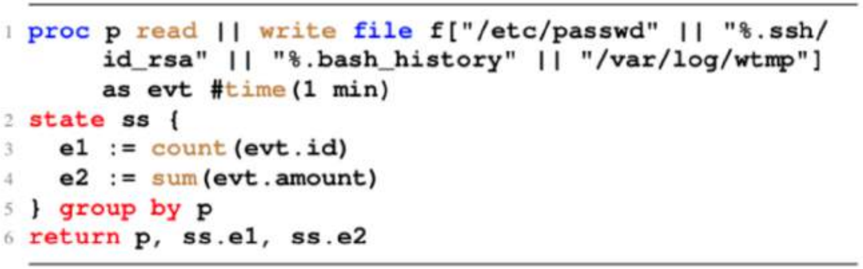


```
agentid = XXX // db server
proc p write ip i as evt #time(10 min)
state[3] ss {
  avg_amount := avg(evt.amount)
} group by p
alert (ss[0].avg_amount > (ss[0].avg_amount + ss[1].
     avg_amount + ss[2].avg_amount) / 3) && (ss[0].
     avg_amount > 10000)
return p, ss[0].avg_amount, ss[1].avg_amount, ss[2].
     avg_amount
```

- **Multievent pattern matching**: match the stream against the event patterns
- **Stateful computation**: compute and maintain states over sliding windows
- **Alert condition checking**: check conditions for triggering alerts
- **Return and filters**: return desired attributes of qualified events

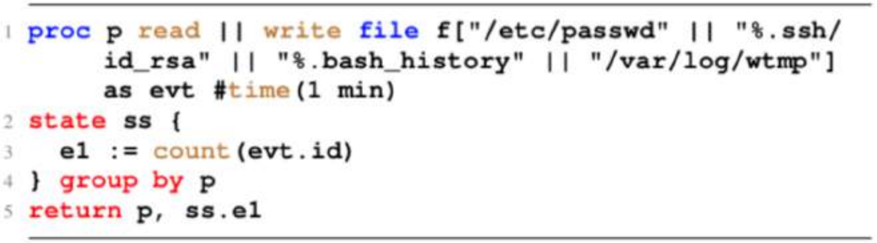
Master-Dependent-Query Scheme

- Challenge: executing multiple concurrent queries incurs considerable overhead
- Key insight: share intermediate execution results among queries (two levels for now: event pattern matching, stateful computation)
  - ➢ Partition concurrent queries into *master-dependent groups*
  - ➢ Only master query has direct access to the stream



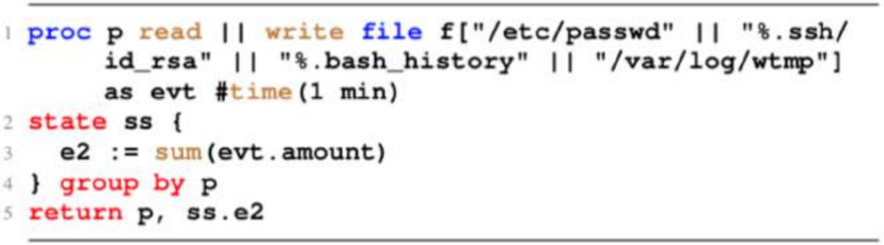
```
proc p read || write file f["/etc/passwd" || "%.ssh/
      id_rsa" || "%.bash_history" || "/var/log/wtmp"]
      as evt #time(1 min)
state ss {
  e1 := count(evt.id)
  e2 := sum(evt.amount)
} group by p
return p, ss.e1, ss.e2
```

Master query



```
proc p read || write file f["/etc/passwd" || "%.ssh/
      id_rsa" || "%.bash_history" || "/var/log/wtmp"]
      as evt #time(1 min)
state ss {
  e1 := count(evt.id)
} group by p
return p, ss.e1
```

Dependent query 1

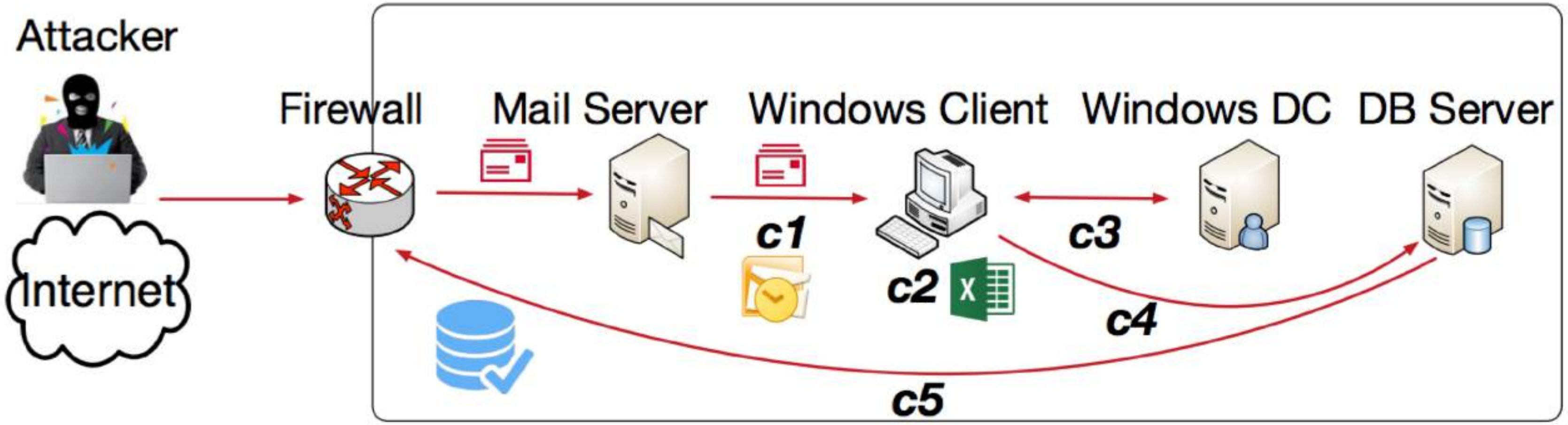


```
proc p read || write file f["/etc/passwd" || "%.ssh/
      id_rsa" || "%.bash_history" || "/var/log/wtmp"]
      as evt #time(1 min)
state ss {
  e2 := sum(evt.amount)
} group by p
return p, ss.e2
```

Dependent query 2

Case Study: Four Major Types of Attacks

- Deploy in NEC Labs of 150 hosts (1.1 TB data; 3.3 billion events; throughput 3750 events/s)
- Deployed server has 12 cores and 128GB of RAM
- 17 queries
    - **APT attack**: *apt-c1, apt-c2, apt-c3, apt-c4, apt-c5, apt-c2-invariant, apt-c5-timeseries, apt-c5-outlier*
    - **SQL injection attack**: *sql-injection*
    - **Bash shellshock command injection attack**: *shellshock*
    - **Suspicious system behaviors**: *dropbox, command-history, password, login-log, sshkey, usb, ipfreq*



Case Study: Execution Statistics

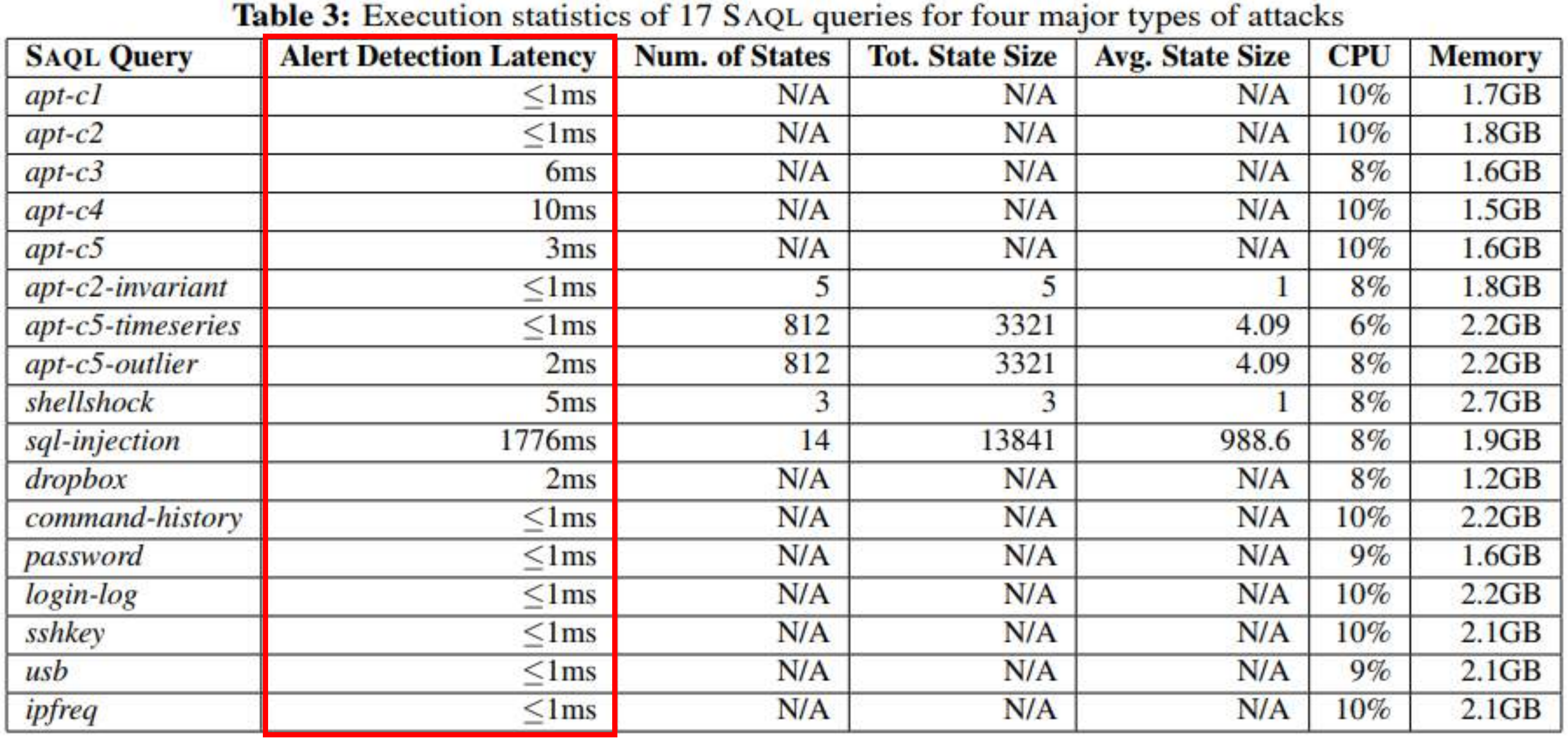
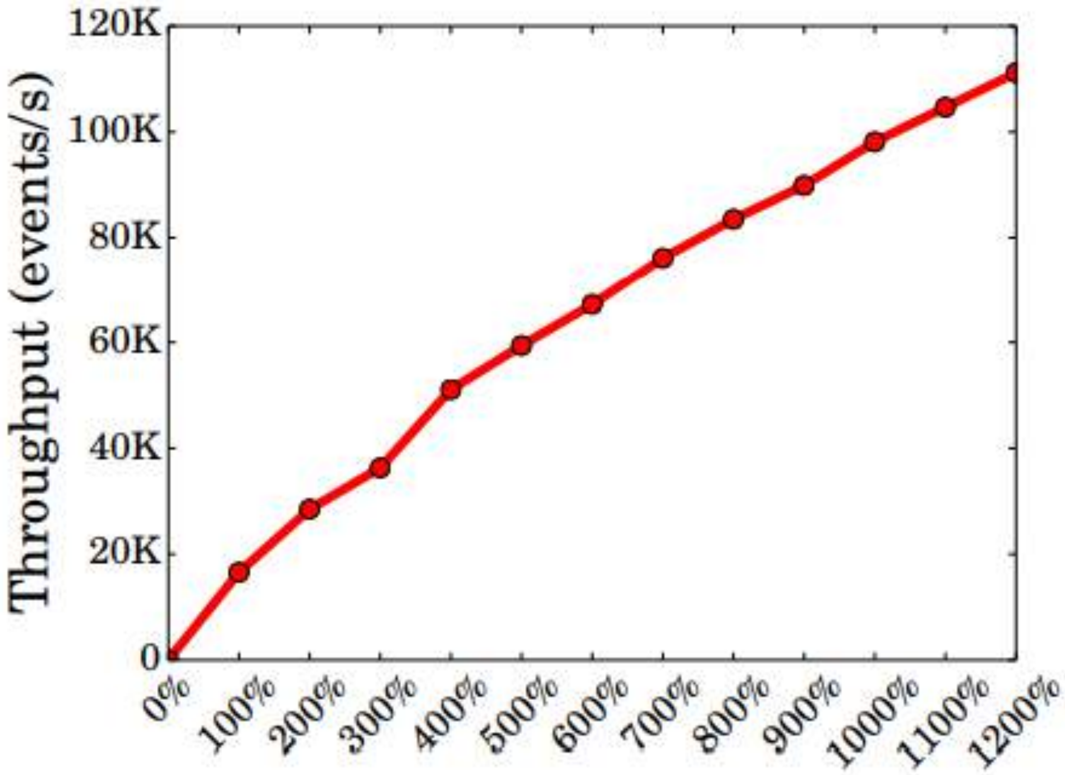


Table 3: Execution statistics of 17 SAQL queries for four major types of attacks

SAQL Query	Alert Detection Latency	Num. of States	Tot. State Size	Avg. State Size	CPU	Memory
apt-c1	≤1ms	N/A	N/A	N/A	10%	1.7GB
apt-c2	≤1ms	N/A	N/A	N/A	10%	1.8GB
apt-c3	6ms	N/A	N/A	N/A	8%	1.6GB
apt-c4	10ms	N/A	N/A	N/A	10%	1.5GB
apt-c5	3ms	N/A	N/A	N/A	10%	1.6GB
apt-c2-invariant	≤1ms	5	5	1	8%	1.8GB
apt-c5-timeseries	≤1ms	812	3321	4.09	6%	2.2GB
apt-c5-outlier	2ms	812	3321	4.09	8%	2.2GB
shellshock	5ms	3	3	1	8%	2.7GB
sql-injection	1776ms	14	13841	988.6	8%	1.9GB
dropbox	2ms	N/A	N/A	N/A	8%	1.2GB
command-history	≤1ms	N/A	N/A	N/A	10%	2.2GB
password	≤1ms	N/A	N/A	N/A	9%	1.6GB
login-log	≤1ms	N/A	N/A	N/A	10%	2.2GB
sshkey	≤1ms	N/A	N/A	N/A	10%	2.1GB
usb	≤1ms	N/A	N/A	N/A	9%	2.1GB
ipfreq	≤1ms	N/A	N/A	N/A	10%	2.1GB

Low detection latency: <2s

Pressure Test



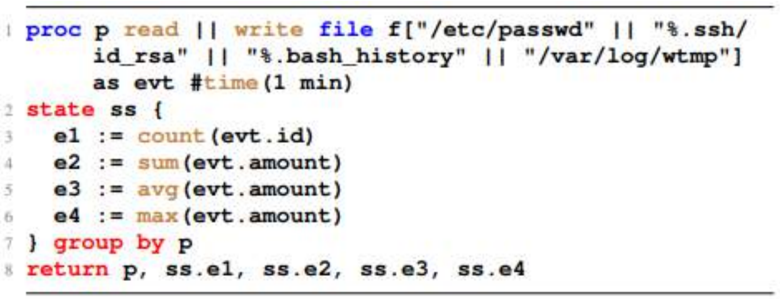
High system throughput: 110,000 events/s; supporting ~4000 hosts

Performance of Concurrent Query Execution

- 64 micro-benchmark queries
  - ➢ Four **attack categories**:
    - ▪ Sensitive file access: */etc/password, .ssh/id_rsa, .bash_history, /var/log/wtmp*
    - ▪ Browsers access files: *chrome, firefox, iexplore, microsoftedge*
    - ▪ Processes access networks: *dropbox, sqlservr, apache, outlook*
    - ▪ Processes spawn: */bin/bash, /usr/bin/ssh, cmd.exe, java*
  - ➢ Four **evaluation categories** for query variations:
    - ▪ Event attribute: 1 attribute -> 4 attributes
    - ▪ Sliding window: 1 minute -> 4 minute
    - ▪ Agent ID: 1 agent -> 4 agents
    - ▪ State aggregation: 1 aggregation type -> 4 aggregation types
  - ➢ 4 queries for each joint category, 64 = 4 * 4 * 4

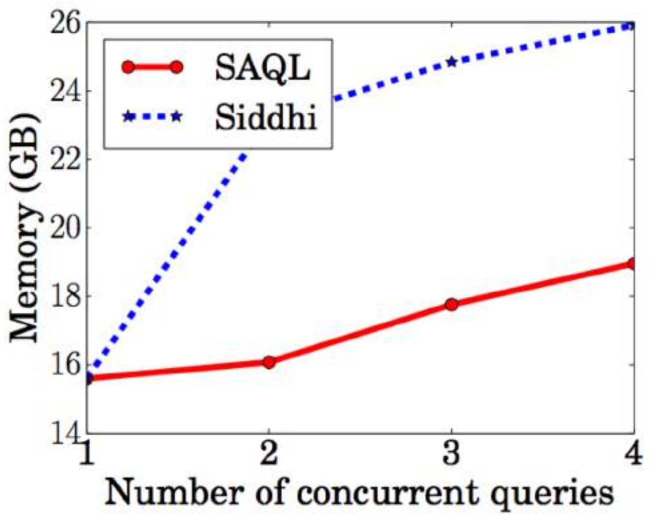
Performance of Concurrent Query Execution

- Example micro-benchmark query for joint category “sensitive file accesses & state aggregation”



```
proc p read || write file f["/etc/passwd" || "%.ssh/
     id_rsa" || "%.bash_history" || "/var/log/wtmp"]
     as evt #time(1 min)
state ss {
   e1 := count(evt.id)
   e2 := sum(evt.amount)
   e3 := avg(evt.amount)
   e4 := max(evt.amount)
} group by p
return p, ss.e1, ss.e2, ss.e3, ss.e4
```

- Memory consumption (MB) w.r.t. number of concurrent queries



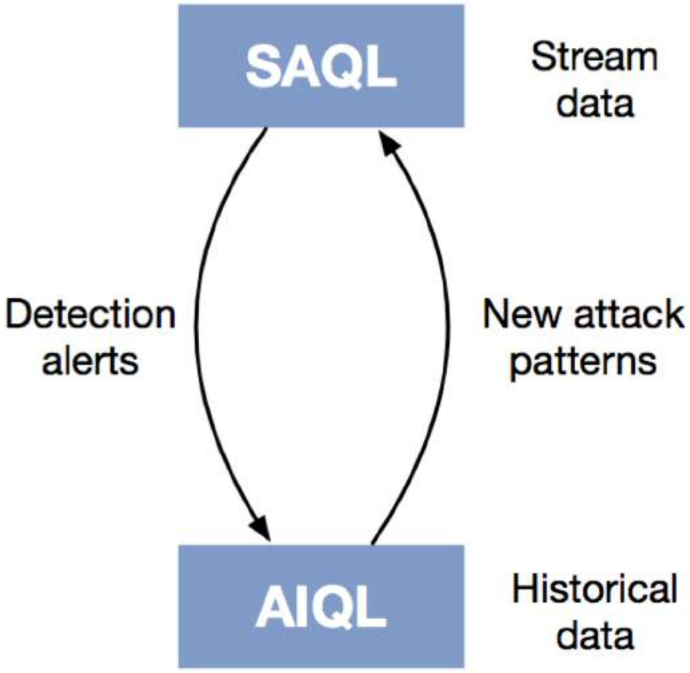
30% average memory saving for all 64 categories

Alert Detection and Investigation

- Historical data is required for alert investigation

- **AIQL (Attack Investigation Query Language) System (*USENIX ATC'18*)**
    - ➢ Data stored in relational databases with efficient indexing
    - ➢ Compatible query language
    - ➢ Leverage domain specifics to speedup the search of complex system event patterns
    - ➢ Project website: https://sites.google.com/site/aiqlsystem/

- Together, SAQL and AIQL work seamlessly for defending against APT attacks



SAQL

Stream data

Detection alerts

New attack patterns

AIQL

Historical data

Conclusion

- **SAQL (Stream-based Anomaly Query Language) System** : enabling timely anomaly detection via querying the real-time stream of system monitoring data
  - ➢ Concisely express four types of anomaly models
  - ➢ Efficient stream management and concurrent query execution based on domain specifics
  - ➢ Project website: https://sites.google.com/site/saqlsystem/

Q & A

Thank you!

CASE WESTERN RESERVE UNIVERSITY EST. 1826

PRINCETON UNIVERSITY

NEC Laboratories America
Relentless passion for innovation