AIQL: Enabling Efficient Attack Investigation from System Monitoring Data

Peng Gao\textsuperscript{1}, Xusheng Xiao\textsuperscript{2}, Zhichun Li\textsuperscript{3}, Kangkook Jee\textsuperscript{3}, Fengyuan Xu\textsuperscript{4}, Sanjeev R. Kulkarni\textsuperscript{1}, Prateek Mittal\textsuperscript{1}

\textsuperscript{1}Princeton University
\textsuperscript{2}Case Western Reserve University
\textsuperscript{3}NEC Laboratories America, Inc.
\textsuperscript{4}Nanjing University
Introduction
The Equifax Data Breach
Impact of Advanced Persistent Threat (APT) Attack

- **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
Advanced Persistent Threat (APT)

- Compromise
- Infection
- Data Exfiltration
- Obtain database Credential
- Privilege Escalation
- Internal Machine
- Gateway
- Domain Controller
- Database
Challenges and Opportunities

Sophisticated and stealthy

- Multiple steps
- Individual step not suspicious enough
- Stealthy for a certain time

Data Exfiltration

Compromise

Obtain database Credential

Privilege Escalation

Infection
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: processes, files, network connections
  ➢ System activities (system events): <subject, operation, object>,
    ▪ e.g., proc p1 read file f1

• Enabling attack investigation via querying the collected data
  ➢ Interactive query: recovering attack steps in multiple hosts
Challenge 1: **Attack Behavior Specification**

- **Multi-step attack**: multiple system activities and their relationships
- **Dependency tracking**: chaining constraints among system activities
- **Abnormal system behavior**: sliding windows and statistical aggregation methods
Challenge 2: **Timely “Big Data” Security Analysis**

- Collect and store system monitoring data for hosts in an organization (~50 GB for 100 hosts per day)
  - Data storage optimization
- Query data in a central server for attack investigation
  - Query execution optimization
AIQL System

- Novel query system for attack investigation
  - Built on top of existing mature tools
    - System-level monitoring tools: auditd, ETW, DTrace
    - Relational databases: PostgreSQL, Greenplum
  - Implementation: ~50,000 lines of Java code

- System Components <- System Optimizations
  - Data Collection and Storage <- Data Model and Storage
  - AIQL Language Parser <- Domain-Specific Query Language
  - Query Execution Engine <- Relationship-based Scheduling
Data Collection and Storage
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>
<thead>
<tr>
<th>Table 1: Representative attributes of system entities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entity</strong></td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>File</td>
</tr>
<tr>
<td>Process</td>
</tr>
<tr>
<td>Network Connection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Representative attributes of system events</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
</tr>
<tr>
<td>Time/Sequence</td>
</tr>
<tr>
<td>Misc.</td>
</tr>
</tbody>
</table>
Data Storage

• Store data in relational databases powered by PostgreSQL
  ➢ Challenges in handling high ingestion rate

• Data storage optimizations
  ➢ Data deduplication and in-memory indexes
    ▪ Entity tables: file, process, network connection
    ▪ Event tables: file event, process event, network event
  ➢ Batch commit
  ➢ Time and space partitioning
  ➢ Hypertable
AIQL (Attack Investigation Query Language)
Multievent AIQL Query

- Multiple system events involved in the attack
- Temporal order of events: e1->e2->e3->e4
Multievent AIQL Query

- Global constraints: e.g., agent ID, time window
- Event pattern: <subject, operation, object>
- Temporal relationship: e.g., enforce the event order
- Attribute relationship: e.g., two events linked by the same entity
- Syntax shortcuts: e.g., context-aware attribute inference
Dependency AIQL Query

- Chain constraints among events
- Dependency tracking for attack causality analysis
Dependency AIQL Query

```
(at "01/01/2017")
forward: proc p1["%/bin/cp%", agentid = 2] ->[write]
    file f1["/var/www/%info_stealer%"]
<-[read] proc p2["%apache%"]
->[connect] proc p3[agentid=3] // tracking across host
->[write] file f2["%info_stealer%"]
return f1, p1, p2, p3, f2
```

- Dependency tracking direction: forward/backward, ->/<-
- Cross host dependency tracking
Anomaly AIQL Query

- Frequency-based anomaly models
Anomaly AIQL Query

- Sliding windows
- Access to history states
- Frequency-based anomaly models: e.g., SMA
AIQL Query Execution Engine
Execution of Multievent Query

- Synthesize a SQL data query for every event pattern
- Schedule the data queries using domain-specific optimizations
Data Query Scheduler

• Leverage **event relationships** for optimizing **search strategies**
  - Prioritize event search based on estimated **pruning power**
  - Prune search space of **related events**

```plaintext
proc p1 write file f1['%pwdump7.exe'|'%libeay32.dll'] as evt2
proc p2 execute file f1 as evt3
```

• Leverage **domain-specific characteristics** of system monitoring data for **parallel search**
  - Time window partition
Time Window Partition

• Uniform time window partition
  ➢ # of sub queries depending on I/O capacity (concurrent read)
• Other potential partition: uniform workload ...

```sql
(at "mm/dd/2017")
agentid = xxx // SQL database server (obfuscated)
proc p1["%cmd.exe"] start proc p2["%sql.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 il[dstip="XXX.129"] as evt4
with evt1 before evt2, evt2 before evt3, evt3 before
evt4
return distinct p1, p2, p3, f1, p4, il
```
Case Study and Evaluation
Case Study: APT Investigation

- **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 runs the database cracking tool to obtain database credentials.
- **c4 Penetration into Database Server**: Attacker drops another malware to open another backdoor.
- **c5 Data Exfiltration**: Attacker dumps the database content and sends the dump back to his host.
Investigation of Step c5

• Deploy anomaly detectors on the database server to detect large amount of data transfer
  ➢ Identify a suspicious IP “XXX.129”

• Construct an anomaly AIQL query to find the processes that access “XXX.129”
  ➢ “sbblv.exe” (p) writes to “XXX.129”

```sql
(at "mm/dd/2017") // date (obfuscated)
agentid = xxx // SQL database server (obfuscated)
window = 1 min, step = 10 sec
proc p write ip i[dstip="XXX.129"] as evt
return p, avg(evt.amount) as amt
group by p
having (amt > 2 * (amt + amt[1] + amt[2]) / 3)
```
Investigation of Step c5

• Find the files accessed by “sbblv.exe”
  ➢ “sbblv.exe” reads from “backup1.dmp” (f1)

```sql
agentid = xx // SQL database server (obfuscated)
proc pl["%sbblv.exe"] read || write file fl as evt1
proc pl read || write ip il[dstip="XXX.129"] as evt2
with evt1 before evt2
return distinct p1, fl, il, evt1.oftype, evt1.access
```

• Creation processes of “backup1.dmp”
  ➢ “sqlservr.exe” (p1) writes to “backup1.dmp”

```sql
agentid = xxx // SQL database server (obfuscated)
proc p1 write file fl["%backup1.dmp"] as evt1
proc pl["%sbblv.exe"] read file fl as evt2
proc p2 read || write ip il[dstip="XXX.129"] as evt3
with evt1 before evt2, evt2 before evt3
return distinct p1, fl, p2, il
```
Investigation of Step c5

- Further verification: “cmd.exe” starts “osql.exe” (OSQL utility)
- Complete AIQL query for c5 (data exfiltration)
End-to-End Efficiency for Case Study

- 27 queries, touching 119 GB of data (422 million system events)

Table 3: Aggregate statistics for case study

<table>
<thead>
<tr>
<th>Attack Step</th>
<th># of Queries</th>
<th># of Evt Patterns</th>
<th>AIQL (s)</th>
<th>PostgreSQL (s)</th>
<th>Neo4j (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>1</td>
<td>3</td>
<td>3.8</td>
<td>3.1</td>
<td>10.8</td>
</tr>
<tr>
<td>c2</td>
<td>8</td>
<td>27</td>
<td>31.0</td>
<td>8038.7</td>
<td>10981.7</td>
</tr>
<tr>
<td>c3</td>
<td>2</td>
<td>4</td>
<td>15.9</td>
<td>15.3</td>
<td>3615.6</td>
</tr>
<tr>
<td>c4</td>
<td>8</td>
<td>35</td>
<td>61.0</td>
<td>10906.7</td>
<td>8150.6</td>
</tr>
<tr>
<td>c5</td>
<td>7</td>
<td>18</td>
<td>58.8</td>
<td>2166.5</td>
<td>4285.4</td>
</tr>
<tr>
<td>All</td>
<td>26</td>
<td>87</td>
<td>170.5</td>
<td>21130.3</td>
<td>27044.1</td>
</tr>
</tbody>
</table>

~3 min 124x slower 157x slower
Conciseness Evaluation

- 19 queries on four major types of attack behaviors, touching 738 GB of data (2.1 billion events)
  - a1-a5: Multi-step attack behaviors
  - d1-d3: Dependency tracking behaviors
  - v1-v5: Real-world malware behaviors (from Virussign)
  - s1-s6: Abnormal system behaviors

Other languages vs. AIQL: 2.4x more constraints, 3.1x more words, and 4.7x more characters
Scheduling Efficiency in Our Optimized Storage

• Efficiency in PostgreSQL
  ➢ 40x speedup

• Efficiency in Greenplum
  ➢ 16x speedup
Conclusion
Conclusion

• **AIQL**, a system for efficient attack investigation from system monitoring data
  - Expressive and concise domain-specific query language
  - Efficient query execution based on domain-specific data characteristics

• All queries, the investigation details, and a system demo video are available in project website
  - [https://sites.google.com/site/aiqlsystem/](https://sites.google.com/site/aiqlsystem/)
  - [https://youtu.be/5v3Tk6Y6YkQ](https://youtu.be/5v3Tk6Y6YkQ)

Q & A
Thank you!