Querying Streaming System Monitoring Data for Enterprise System Anomaly Detection

Peng Gao[1] Xusheng Xiao[2] Ding Li[3] Kangkook Jee[4] Haifeng Chen[3] Sanjeev R. Kulkarni[5] Prateek Mittal[5]
[1]*UC Berkeley* [2]*Case Western Reserve University* [3]*NEC Labs America* [4]*UT Dallas* [5]*Princeton University*
penggao@berkeley.edu xusheng.xiao@case.edu {dingli,haifeng}@nec-labs.com kangkook.jee@utdallas.edu {kulkarni,pmittal}@princeton.edu

***Abstract*—The need for countering Advanced Persistent Threat (APT) attacks has led to the solutions that ubiquitously monitor system activities in each enterprise host, and perform timely abnormal system behavior detection over the stream of monitoring data. However, existing stream-based solutions lack explicit language constructs for expressing anomaly models that capture abnormal system behaviors, thus facing challenges in incorporating *expert knowledge* to perform *timely anomaly detection* over the large-scale monitoring data. To address these limitations, we build SAQL, a novel stream-based query system that takes as input, a real-time event feed aggregated from multiple hosts in an enterprise, and provides an anomaly query engine that queries the event feed to identify abnormal behaviors based on the specified anomaly models. SAQL provides a domain-specific query language, *Stream-based Anomaly Query Language (*SAQL*)*, that uniquely integrates critical primitives for expressing major types of anomaly models. In the demo, we aim to show the complete usage scenario of SAQL by (1) performing an APT attack in a controlled environment, and (2) using SAQL to detect the abnormal behaviors in real time by querying the collected stream of system monitoring data that contains the attack traces. The audience will have the option to interact with the system and detect the attack footprints in real time via issuing queries and checking the query results through a command-line UI.**

I. Introduction

Advanced cyber attacks and data breaches plague even the most protected businesses [7], [5]. Similar attacks, especially in the form of Advanced Persistent Threats (APTs), are being commonly observed. These attacks consist of *a sequence of steps* across *many hosts* that exploit different types of vulnerabilities to compromise security. To counter these attacks, approaches based on *ubiquitous system monitoring* have emerged as an important solution for *monitoring system activities and actively detecting possible abnormal system behaviors* [13], [15], [10], [9]. System monitoring observes system calls at the kernel level to collect system-level events that record interactions among system entities (*e.g.,* processes, files, and network connections). Collection of system monitoring data enables security analysts to detect abnormal system behaviors by *continuously searching for anomalies from the streaming data* [8], [14].

Fighting against advanced attacks such as APTs is a time-critical mission. As such, there is a strong need for a *real-time* anomaly detection system that can find a "needle in a haystack" from system monitoring data for preventing additional damage and performing system recovery. However, there are two major challenges for building such system to support effective and timely anomaly detection: (1) *Expert Knowledge Incorporation*: Advanced attacks typically involve multiple steps exploiting various types of vulnerabilities. Besides, models derived from data have been increasingly used in detecting various types of abnormal behaviors [14]. System administrators, security analysts, and data scientists have extensive domain knowledge about the enterprise, including the expected system behaviors. Fighting against such attacks requires the system to provide a unified interface for expressing a broad range of anomaly models while *incorporating the domain knowledge from experts*; (2) *Timely Big-Data Analytics*: System monitoring produces huge amount of daily logs (∼50GB for 100 hosts per day) [13], [15]. This requires the system to provide *efficient* real-time data analytics.

Unfortunately, none of the existing systems [11], [12], [6], [1] provide a comprehensive solution that addresses both of these inherent challenges. Existing anomaly detection systems focus on building models for specific anomalies based on extracted features, rather than providing a unified interface for expressing a broad range of anomaly models via incorporating the expert knowledge. Existing stream-based query systems are designed to work with general-purpose data streams, and lack explicit language constructs for expressing various anomaly models for our particular problem domain. Furthermore, to support multiple concurrent queries that access different attributes of the data, these systems have to make multiple copies of the data for the queries, and thus is not efficient in handling the big data collected from system monitoring.

To address these challenges, we build SAQL [9], a stream-based query system that enables security analysts to perform real-time abnormal system behavior detection via querying the stream of system monitoring data. SAQL takes as input a real-time event feed aggregated from multiple hosts in an enterprise, and provides an anomaly query engine that queries the event feed to identify abnormal system behaviors based on the specified anomaly models. To facilitate the task of expert knowledge incorporation, SAQL provides a domain-specific query language, *Stream-based Anomaly Query Language (*SAQL*)*, that uniquely integrates a series of critical primitives for expressing a broad range of anomaly models. In particular, SAQL provides (1) the syntax of event patterns for specifying relevant system activities and their relationships, which facilitates the specification of *rule-based anomaly models*; (2) the constructs for *sliding windows* and *stateful computation* that allow stateful anomaly models to be computed in each sliding window over the data stream. These

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00167

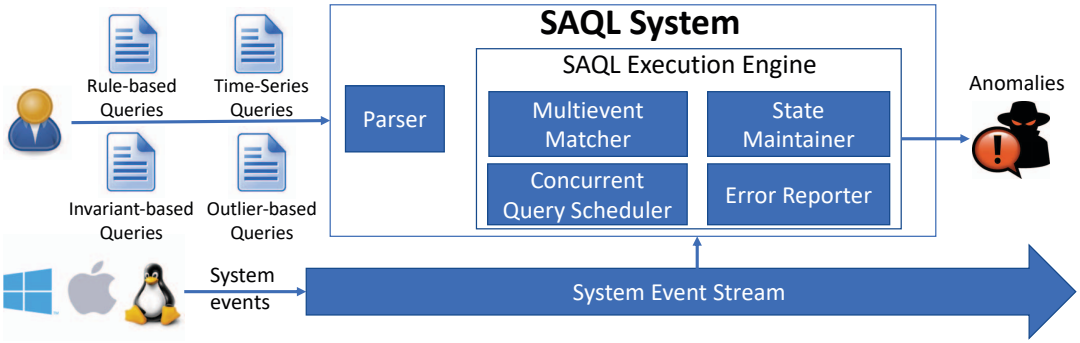

Fig. 1: The architecture of SAQL

constructs facilitate the specification of *time-series anomaly models*, *invariant-based anomaly models*, and *outlier-based anomaly models*, which lack support from existing stream-based query systems. To facilitate the task of efficiently handling concurrent queries, SAQL employs a master-dependent-query scheme that groups semantically compatible queries to minimize the data copies of the stream. Our anomaly query engine leverages the domain-specific characteristics of the system monitoring data and the semantics of the queries to efficiently schedule the execution.

We have deployed the SAQL system in NEC Labs America comprising 150 hosts and made a demo video [4]. In our demo, we aim to show the complete usage scenario of SAQL. To achieve this goal, we perform an APT attack in a controlled environment (for protecting the normal business) that exploits multiple vulnerabilities in the system and exfiltrates sensitive data from database server. The system monitoring data that contains the attack traces is collected by our data collection agents and sent to the central server, forming an event stream. We constructed a set of SAQL queries in advanced for detecting the attack behaviors and deployed them over the stream. These SAQL queries will continuously monitor the stream and report the alerts in real time as we perform the attack. To easily reproduce the attack data for showcasing different queries, we additionally store the data in databases and have built a stream replayer to replay the data from databases as a data stream. The audience will have the option to interact with the system and detect the attack footprints via issuing queries and checking the query results through a command-line UI.

II. The SAQL System Architecture

Figure 1 shows the architecture of the SAQL system. SAQL takes an input query from the user that specifies certain attack behaviors to be detected, executes the query by checking the specified behaviors against the system event stream, and reports the detection alerts once there exist matches.

A. Data Collection

System monitoring records kernel-level interactions among system entities as system events. Each of the recorded event occurs on a particular host at a particular time, thus exhibiting strong spatial and temporal properties. Following the established convention [13], [15], [9], [10], in our data model, we consider *system entities* as files, processes, and network connections. We consider a *system event* as the interaction between two system entities represented as ⟨*subject, operation, object*⟩ (SVO). Subjects are processes originating from software applications (*e.g.,* Firefox), and objects can be files, processes, and network connections. We categorize system events into three types according to their objects, namely *file events*, *process events*, and *network events*.

We build data collection agents based on mature system monitoring frameworks: auditd for Linux, ETW for Windows, and DTrace for MacOS. Our agents are deployed across servers, desktops, and laptops in the enterprise to collect system auditing events from kernels. The collected events with critical security-related attributes (e.g., file name, process executable name, PID, IP, port; more details in [9]) are sent to the central server, forming an event stream.

B. SAQL Query Language

We build the SAQL language using ANTLR 4. Our language uniquely integrates a series of critical primitives for concisely expressing four major types of anomaly models.

1) Rule-based Anomaly Model: SAQL provides explicit constructs to specify system entities/events, attribute constraints, and event temporal/attribute relationships. This facilitates the specification of rule-based anomaly models to detect known attack behaviors or enforce enterprise-wide security policies. Query 1 shows a SAQL query that detects the data exfiltration from database server: the attacker leverages OSQL utility (`osql.exe`) to dump the database content (`backup1.dmp`) and then runs a malware (`sbblv.exe`) to send the dump back to his host (`xxx.129`). Four event patterns are declared (Lines 2-5) with a global constraints (Line 1), a temporal relationship (Line 6), and an implicit attribute relationship (Lines 3-4 specify the same `f1` in both events). Desired attributes of matched events are returned (Line 7) with context-aware syntax shortcuts adopted (*i.e.,* `p1` → `p1.exe_name`).

```
1 agentid = xxx // SQL database server (obfuscated)
2 proc p1["%cmd.exe"] start proc p2["%osql.exe"] as evt1
3 proc p3["%sqlservr.exe"] write file f1["%backup1.dmp"] as
      evt2
4 proc p4["%sbblv.exe"] read file f1 as evt3
5 proc p4 read || write ip i1[dstip="XXX.129"] as evt4
6 with evt1 -> evt2 -> evt3 -> evt4
7 return distinct p1, p2, p3, f1, p4, i1 // p1 -> p1.exe_name
      , i1 -> i1.dstip, f1 -> f1.name
```

Query 1: A rule-based SAQL query

2) Time-Series Anomaly Model: SAQL provides explicit constructs for sliding windows and stateful computation that allow stateful anomaly models to be computed in each sliding window over the stream. These constructs lay the foundation for specifying advanced anomaly models (*i.e.,* time-series anomaly models, invariant-based anomaly models), which lack support from existing stream-based query systems [11], [12], [6], [1]. Query 2 shows a SAQL query that specifies a time-series anomaly model to monitor the network usage of each application and raises an alert when the network usage is abnormally high. It specifies a 10-minute sliding window (Line 1), collects the amount of data sent through network within each window (Lines 2-4), and computes the moving average to detect spikes of network data transfers (Line 5). In the query, `ss[0]` represents the state of the current window while `ss[1]` and `ss[2]` represent the states of the two past windows respectively (`ss[2]` occurs earlier than `ss[1]`).

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

Query 2: A time-series SAQL query

3) Invariant-based Anomaly Model: SAQL provides explicit constructs for learning invariants of system behaviors under normal operations and using the learned invariants to detect later violations. This facilitates the specification of invariant-based anomaly models. Query 3 shows a SAQL query that specifies a 10-second sliding window (Line 1), maintains a set of child processes spawned by the Apache process (Lines 2-4), uses the first ten windows to train the invariant (Lines 5-8), and detects unseen child processes spawned by Apache (Line 9).

```
1 proc p1["%apache.exe"] start proc p2 as evt #time(10 s)
2 state ss {
3   set_proc := set(p2.exe_name)
4 } group by p1
5 invariant[10][offline] {
6   a := empty_set // invariant init
7   a = a union ss.set_proc //invariant update
8 }
9 alert |ss.set_proc diff a| > 0
10 return p1, ss.set_proc
```

Query 3: An invariant-based SAQL query

4) Outlier-based Anomaly Model: SAQL provides explicit constructs for grouping system behaviors together to detect outliers. This facilitates the specification of outlier-based anomaly models. Query 4 shows a SAQL query that specifies a 10-minute sliding window (Line 2), computes the amount of data sent through network by the `sqlservr.exe` process for each outgoing IP address (Lines 3-5), and identifies the outliers using DBSCAN clustering (Lines 6-7) to detect the suspicious IP that triggers the database dump. Note that Line 6 specifies which information of the state forms a comparison point and how the "distance" among these points should be computed (`"ed"` represents Euclidean Distance).

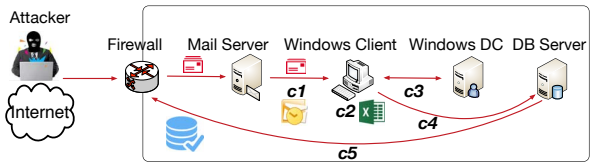

Fig. 2: Demonstration setup for the APT attack

```
| $ ./run.sh
Input the SAQL query:

proc p read || write file f as evt#timeBatch(1 m)
state ss {
        freq := distinctcount(f)
} group by p
cluster(points=all(ss.freq), distance="ed", method="DBSCAN(5, 10)")
alert cluster.outlier && ss.freq > 100
return p, ss.freq as freq


State SAQL query
log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
rbId: 6c50fd69x90d6x40b3x9892x893b8fb9b9e6
###window 1 start###
p_exe_name:C:\PROGRAM FILES (X86)\SYMANTEC\SYMANTEC ENDPOINT PROTECTION\12.1.7369.6900.105\BIN\CCSVCHST.EXE     freq:368.0
p_exe_name:C:\PROGRAM FILES (X86)\GOOGLE\CHROME\APPLICATION\CHROME.EXE  freq:1262.0
p_exe_name:C:\WINDOWS\SYSTEM32\RUNTIMEBROKER.EXE        freq:124.0
p_exe_name:(System)     freq:240.0
p_exe_name:C:\WINDOWS\SYSTEM32\SVCHOST.EXE      freq:282.0
p_exe_name:/usr/sbin/cron       freq:145.0
p_exe_name:/usr/lib/sm.bin/sendmail     freq:14935.0
###window 1 end###
###window 2 start###
p_exe_name:C:\PROGRAM FILES (X86)\SYMANTEC\SYMANTEC ENDPOINT PROTECTION\12.1.7369.6900.105\BIN\CCSVCHST.EXE     freq:223.0
p_exe_name:C:\PROGRAM FILES\SYMANTEC ENDPOINT PROTECTION\12.1.7369.690.105\BIN\CCSVCHST.EXE     freq:319.0
p_exe_name:C:\PROGRAM FILES (X86)\GOOGLE\CHROME\APPLICATION\CHROME.EXE  freq:613.0
p_exe_name:(System)     freq:542.0
p_exe_name:C:\WINDOWS\SYSTEM32\SVCHOST.EXE      freq:216.0
p_exe_name:/usr/share/mailspring/mailspring     freq:109.0
p_exe_name:/usr/lib/sm.bin/sendmail     freq:37831.0
###window 2 end###
```

Fig. 3: Command-line UI of the SAQL system

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

Query 4: An outlier-based SAQL query

C. SAQL Query Execution Engine

We build the SAQL system upon Siddhi CEP [6] so that our system can leverage Siddhi's mature mechanisms to manage the event stream. Given an input SAQL query, the multievent matcher matches the events in the stream against the event patterns specified in the query. If the query involves stateful computation, the state maintainer maintains the states of each sliding window computed from the matched events. To efficiently handle the execution of multiple concurrent queries, the concurrent query scheduler employs a master-dependent-query scheme. In the scheme, concurrent queries are divided into groups based their semantic compatibilities, with each group having a master query and several dependent queries. The scheme enforces that the queries in a group will share a single copy of the stream data for execution. Only master queries have direct access to the data stream, and the execution of the dependent queries leverages the intermediate execution results of their master query. In this way, unnecessary data copies of the stream can be significantly reduced. The error reporter reports the errors during the query execution. Alerts are generated when the alert conditions specified in the query are matched by the event stream.

III. Demonstration Outline

Demonstration Setup for The APT Attack. We deployed SAQL in NEC Labs America comprising 150 hosts. The purpose of our demo is to illustrate the complete usage scenario of SAQL and showcase its superiority in enabling timely abnormal system behavior detection. To achieve this goal, we perform an APT attack in a controlled environment (Figure 2) using known exploits. The APT attack consists of five steps as follows:

c1 Initial Compromise: The attacker sends a crafted email to the victim. The email contains an Excel file with a malicious macro embedded.

c2 Malware Infection: The victim opens the Excel file through the Outlook client and runs the macro, which downloads and executes a malicious script (CVE-2008-0081 [2]) to open a backdoor for the attacker.

c3 Privilege Escalation: The attacker enters the victim's machine through the backdoor, scans the network ports to discover the IP address of the database, and runs the database cracking tool (`gsecdump.exe`) to steal the credentials of the database.

c4 Penetration into Database Server: Using the credentials, the attacker penetrates into the database server and delivers a VBScript to drop another malicious script, which creates another backdoor.

c5 Data Exfiltration: With the access to the database server, the attacker dumps the database content using `osql.exe` and sends the data dump back to his host.

Construction of SAQL Queries. We constructed 8 SAQL queries (more details in [3]) in advance for detecting the attack behaviors. For each attack step, we construct a rule-based SAQL query by leveraging the knowledge of the attack. Furthermore, we construct three advanced anomaly queries, assuming no knowledge of the attack details:

- We construct an invariant-based anomaly query to detect the scenario where Excel executes a malicious script that it has never executed before (*i.e.,* step *c2*): The invariant contains all unique processes started by Excel in the first 100 sliding windows. New processes that deviate from the invariant are reported as alerts.
- We construct a time-series anomaly query based on SMA to detect the scenario where abnormally high volumes of data are exchanged via network on the database server (*i.e.,* step *c5*): For every process on the database server, this query detects the processes that transfer abnormally high volumes of data to the network.
- We construct an outlier-based anomaly query to detect processes that transfer high volumes of data to the network (*i.e.,* step *c5*): The query detects such processes through peer comparison based on DBSCAN.

Demonstration Procedure. We start the demonstration by constructing and executing the SAQL queries using a command-line UI (Figure 3). As we perform the attack, the SAQL queries will continuously monitor the stream and report the alerts when the abnormal attack behaviors are detected.

Fig. 4: Stream replayer

To easily reproduce the streaming attack data for showcasing different queries, we additionally store the data in databases and have built a stream replayer to replay the data from databases as a data stream. Our stream replayer has a web-based UI (Figure 4) that lets us choose the hosts and the start/end time to replay the system monitoring data.

IV. Conclusion

We have presented SAQL, a novel system for detecting abnormal system behaviors in enterprises via querying streaming system monitoring data. SAQL provides an expressive domain-specific language to express a wide range of anomaly models.

Acknowledgement. This work was supported in part by DARPA N66001-15-C-4066. Any opinions, findings, and conclusions made in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Apache Flink. https://flink.apache.org/.
[2] CVE-2008-0081. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0081.
[3] Demo SAQL queries. https://goo.gl/Xwrfsf.
[4] Demo video of SAQL. https://youtu.be/3S7D5jVoR2c.
[5] The Equifax data breach. https://www.ftc.gov/equifax-data-breach.
[6] Siddhi. https://github.com/wso2/siddhi.
[7] The Target data breach. https://goo.gl/2awnKE.
[8] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.
[9] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal. SAQL: A stream-based query system for real-time abnormal system behavior detection. In *USENIX Security*, pages 639–656, 2018.
[10] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal. AIQL: Enabling efficient attack investigation from system monitoring data. In *USENIX ATC*, pages 113–126, 2018.
[11] M. N. Garofalakis, J. Gehrke, and R. Rastogi, editors. *Data stream management - processing high-speed data streams*. Springer, 2016.
[12] B. Hossbach and B. Seeger. Anomaly management using complex event processing: Extending data base technology paper. In *EDBT*, pages 149–154, 2013.
[13] S. T. King and P. M. Chen. Backtracking intrusions. In *SOSP*, pages 223–236, 2003.
[14] C. Kruegel, F. Valeur, and G. Vigna. *Intrusion detection and correlation - challenges and solutions*, volume 14. Springer, 2005.
[15] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang. High fidelity data reduction for big data security dependency analyses. In *CCS*, pages 504–516, 2016.