An Event-based Data Model for Granular Information Flow Tracking [*]

Joud Khoury Timothy Upthegrove Armando Caro Brett Benyo Derrick Kong
Raytheon BBN Technologies

Abstract

We present a common data model for representing causal events across a wide range of platforms and granularities. The model was developed for attack provenance analysis under the DARPA Transparent Computing program. The unified model successfully expresses data provenance across a range of granularities (e.g., object or byte level) and platforms (e.g., Linux and Android, BSD, and Windows). This paper describes our experience developing the common data model, the lessons learned, and performance results in controlled lab experiments.

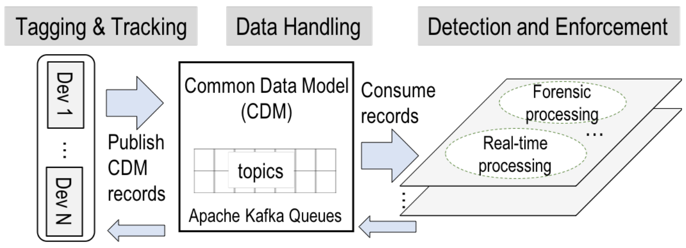


Figure 1: Transparent computing technology pipeline. Multiple instrumented devices publish CDM tags using the common data model API for consumption by multiple analysis and enforcement engines.

1 Introduction

Modern cyber threats, such as the Advanced Persistent Threat (APT), are becoming increasingly hard to detect and counter. These sophisticated programs can lay dormant for long durations and use stealthy long-term reconnaissance and subversion to achieve their goals, such as information exfiltration. DARPA's Transparent Computing (TC) program aims at early detection of APTs and root cause analysis by making otherwise opaque enterprise systems transparent [7]. Transparency is achieved through granular tagging and tracking of causal relationships among programs and data across the enterprise's communication/computation planes, assembly of these dependencies into end-to-end behaviors, and reasoning over the behaviors both forensically and in real-time.

Core TC technologies were organized into a three layer pipeline as shown in Figure 1. The first layer in the pipeline is tagging and tracking, where a diverse set of enterprise and mobile platforms are instrumented to produce streams of granular causal events, tags, and metadata. The event streams are passed to the data handling layer for normalization and storage using a common data model and API. The streaming data is then consumed by the detection and enforcement layer for forensic analysis and real-time APT detection, as well as proactive enforcement of protection policies.

Tagging and tracking techniques build causal relationships among activities, carefully balancing tracking granularity against speed of detection. A large breadth of technologies exist, some of which are coarse-grained such as those based on audit logging systems (e.g., [9, 12]) while others are finer-grained such as with information flow taint tracking systems (e.g., [2, 5, 6]). These technologies can track the flow of data throughout the system at arbitrary granularity all the way from sources to sinks. Technologies also differ by platform (e.g., desktop, server, mobile, embedded), operating system, and layer of the software stack.

Real-time detection and forensic analysis technologies consume the provenance events produced by the tagging and tracking systems, generally building some form of attack provenance graphs and reasoning over them at various timescales to detect policy violations and perform root cause analysis [10, 11]. Accurate provenance information is key to understanding how data flows through a device, which is especially critical for detecting and understanding APTs.

[*]Distribution "A", Approved for Public Release, Distribution Unlimited. This research was developed with funding from the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA), under contract No. FA8750-C-15-7559. The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s)and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

This paper is primarily focused on the Common Data Model (CDM) representation (we refer the reader to [2, 5, 6, 9, 12] for details on the tagging and tracking technologies, and to [10, 11] for details on the detection technologies, all of which directly influenced the CDM design). We identified three main requirements from such a unified representation. First, the CDM should be able to capture metadata about both the data and control planes with high fidelity and at various granularities across platforms. Second, it should be natural for expressing the information produced by the tagging and tracking technologies without adding undue burden on them, for example, by requiring them to capture data they are not meant to capture or to perform unnatural data conversions. Third, it should contain all the semantics needed to enable real-time detection and forensic analysis. The CDM makes minimal assumptions about the data so as not to limit the evolving detection and analysis algorithms. Empowered with the CDM event streams, the different detection and analysis algorithms can each determine how to semantically unify or prune the data to fit its processing needs.

We designed the CDM through consensus with the broader TC team, starting by carefully accounting for the needs of the detection and analysis technologies in context of the data produced by the tagging and tracking technologies. The design evolved over a four year time period as the technology teams gained hands-on experience with it. The CDM encodes events among system entities (principals, subjects, and objects) and granular information flow dependencies among them (e.g., if process reads four files and writes a buffer to a socket, granular tags allow tracking which of the four files contributed to the buffer). Granular information flow among entities enables the APT detection and analysis technologies to reconstruct the causal relationships among the attacker's actions with unprecendented fidelity [10].

We initially considered using and/or extending W3C PROV [4], which is a data model designed for representing provenance. PROV is not well suited for achieving the aforementioned goals of TC for three main reasons. Events in CDM are first class citizens that have rich and complex semantics that cannot be accurately captured using PROV's simple predefined relations. Second, the CDM design adds granular *tag*s for representing information flow at unprecedented fidelity and precision. This is critical for APT detection given the long duration of time over which APT activities unfold. Using W3C PROV would only allow coarse grained tracking of data dependencies, which over time, results in significant loss of accurate provenance. Finally, events in CDM can have arbitrary number of relations to subjects and objects while PROV mainly supports binary relations further restricting expressiveness. We note that detection and analysis technologies can directly convert CDM to W3C PROV by mapping each event to a set of PROV relations. Such conversion can lose information.

This paper presents the following contributions:

1. A common data model and a rationale for its design (§2). The design evolved through consensus among all TC performers over the course of four years.

2. An open source implementation of the model (§3) using Apache Avro [13], along with several test clients for serializing it, and a summary of their performance. The source is publicly available [3].

Open datasets exercising the CDM's ability to encode provenance across a diverse set of platforms and attack scenarios can be found at [1, 8]. The datasets include diverse and detailed example attack scenarios and their descriptions, their CDM encodings, and tools for processing and visualization.

2 Common Data Model

We describe the semantics and syntax of the CDM used for capturing tagged event streams produced by a wide variety of tagging and tracking technologies, as demonstrated in [1, 8]. There are six core entities in the model: *Hosts*, *Principals*, *Subjects*, *Events*, *Objects*, and *ProvenanceTags*. Figure 2 shows these entities and their relationships (see the latest CDM version for the full specification and for a detailed description of each of the entities [3]).

Hosts and Principals A *Host* represent a host machine or node in a network on which principals and objects reside, subjects execute and events occur. A *Principal* represents a local user that owns objects and process subjects.

Subjects *Subject*s represent execution contexts and include mainly threads and processes. They can be more granular and can represent other execution boundaries such as functions and blocks if needed. For example, a function within a thread within a process can be represented as three subjects where the function's *parentSubject* attribute is the thread, and the thread's *parentSubject* attribute is the process. A process' *parentSubject* attribute would be set to the parent process that spawned it. Here the function subject is an execution boundary that is more granular than the thread.

Events *Event*s represent actions executed on behalf of subjects. Events may be represented at various granularities. Events include system calls such as those emitted by audit subsystems (e.g., Windows ETW, Linux auditd, BSD dtrace), function calls at different layers, instruction executions, or even more abstract notions representing a "blind" execution such as black boxes that are not instrumented (more shortly). Events are the core entity in the model and they are the main abstraction for representing information flow between objects and subjects, conceptually represented as a directed edge between subject and object(s) or vice-versa. Events are assumed to be atomic so there is no direct relationship between events.

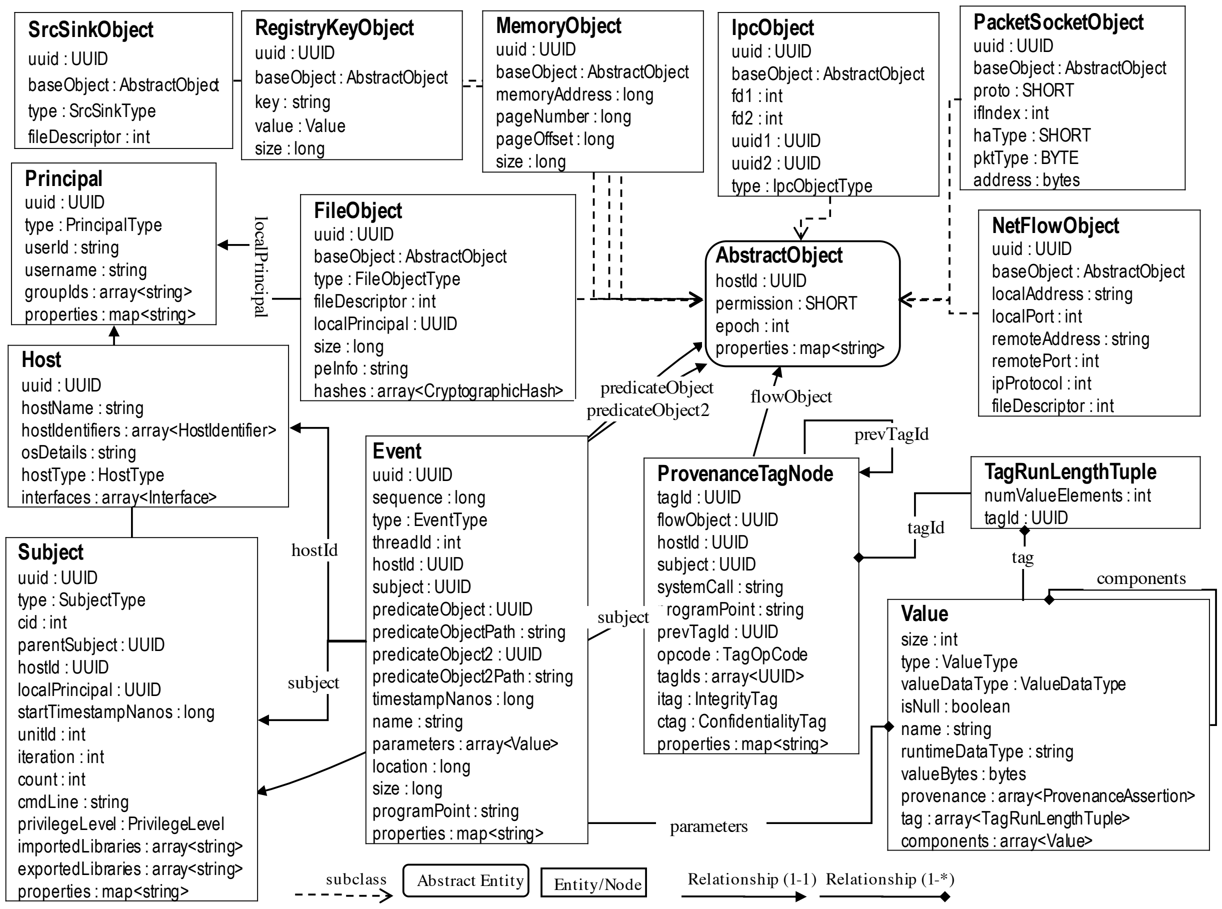


Figure 2: An entity relationship diagram showing the main entities in the CDM and their relationships. Labels over the relationship arrows reflect the corresponding attribute in an entity, and are simply added for readability.

Instead, events are related to other events through the affected subjects and objects. We defined many types of events, mostly corresponding to system calls, see [3].

Events can have different granularities but they are still atomic. For example, a function boundary may execute many atomic events within it, hence the function entry call would be captured as an event and so would the function exit call. The function boundary itself may be captured as a subject in this case if needed. If the function boundary is not instrumented however (black box), the function execution may still be captured as a "blind" event that relates the input and output subjects and objects (more on blindness shortly).

The event *sequence* represents the logical order of the event relative to other events within the same execution thread. This could be the logical time, or if timestamps are accurate enough, sequence might be inferred from timestamps but that is not guaranteed which is why we kept sequence.

Objects *Object*s, in general, represent data sources and sinks which could include sockets, files, registry keys, memory and variables, and any data in general that can be an input and/or output to an event. This model expands the definition of objects to explicitly capture the flow of data inputs and outputs to and from events. We initially decided to treat event arguments as first class objects (instead of attributes of the event entity) where each object may have its own provenance again explicitly showing the flow from inputs to outputs. However we backed away from this later on as discussed shortly.

We identified some key object attributes. The object *timestamp* is the creation time. The file *url* is its local or remote path/location. Files have version numbers. As files get updated, a new file object is created with a new version number.

Transient data: One of the main discussion points was whether we should model arguments as objects or whether we should explicitly distinguish transient data (e.g., arguments) from more persistent data (e.g., files). We agreed that it makes more sense to distinguish the two mainly because they have very different attributes (arguments don't have ownership and path attributes and have a value attribute). Initially, we decided to model a value as a first class entity, just like objects, that can have tags (discussed in §2) and affect (and are affected by) events. We later decided (in version 0.7) to make values attributes of events instead of first class entities since values are one time constructs that only show up with the event and are not referenced afterwards during execution. Tagging and tracking systems also do not track at the level of a value. In addition, treating a value as a first class entity requires that we assign a unique ID to each value which does not scale since values can have byte granularity. As a result, we decided to make values attributes of the events.

Granularity and types: We discussed representing objects at various granularities. To explicitly differentiate between different types of objects, we also decided to subclass the abstract object into *File*, *NetworkFlow*, *Memory*, *PacketSocket*, *RegistryKey*, and *IPC* objects instead of modeling those using a type attribute of the object entity. This is mainly to keep the model clean given the difference in attribute sets among the object types. We include a *RegistryKey* object to model windows registry key store. Finally, we also discussed whether we should model mobile phone sensors (GPS, camera, gyro, etc.) as a separate entity from Object, called Resource. One observation here is that the sensing subsystem in Android for example might have enough differences to warrant this distinction. The counter argument is that Android uses linux underneath and one might be able to get away with an Object abstraction for all such data devices. We decided to only distinguish a few object types (those mentioned above) and everything else uses the generic *SrcSink* object which is typed (e.g., sensors, camera, etc.).

Provenance and Tags We explicitly model the relations between objects and subjects using the events as relationships that connect subjects and objects. These edges represent provenance (control-flow) directly in the graph. The complementary more granular approach for capturing data-flow uses tags, encoded with *ProvenanceTagNode*.

The primary motivation for tags is their ability to capture data flows at a finer granularity and with increased precision, which was important to enable granular tagging and tracking (e.g., [5]) and analysis (e.g., [10]) technologies. Specifically, tags can capture increased precision of information flow from a (strict) subset of inputs represented in a causality graph. For instance, a subject may read from 100 files, and then write one file. Rather than reporting that the output depends on these 100 inputs, a tagging and tracking system can use tags to indicate that the output depends only on the 10th and the 97th input files. In addition, tags can capture the nature of dependence using tag operations.

A provenance tag defines source dependence on specific data sources (inputs). A tag identifier is typically bound to a source and used by the tracking system to capture dependence on this source input. The *ProvenanceTagNode* defines one step of provenance for a value (i.e., one read from a source or write to a sink), a reference to the previous provenance tag of the value (*prevTagId*), and the tag operation that resulted in the tagId of this tag (*opcode*). This graph of tag dependencies provides fine-grained data flow tracking stitching together all the contributing events. Tags are directly associated with event parameters i.e., with values, as attributes of the *Value* entity for fine grained taint tracking. Each value can have an array of tags associated with the bytes that make up the value.

For example, consider program A that takes a picture and writes it to file F1. Then program B reads the image data from file F1, augments it with GPS location data, and writes the result to file F2. At the time of a write, the provenance graph can be emitted showing the granular dependence of the data in F2 on both sources, the camera and the GPS. The provenance tag of the data written to F2 references a union of the two source tags, and the previous tag for each of those references the read events that read the data from the sources, showing the full (both control and data) flow.

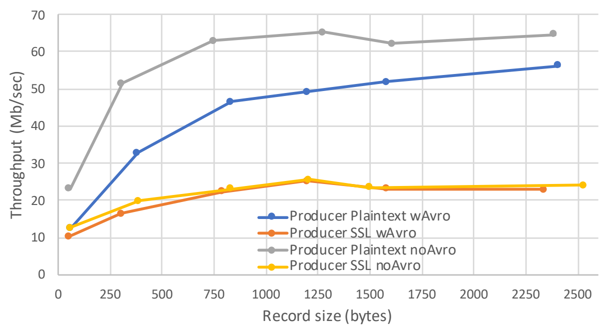
Control Records The *TimeMarker* is used to delineate time periods in a data stream to help consumers know their current read position in the data stream. The *EndMarker* record marks the end of a data stream.

3 Implementation and Performance Results

We implemented a versioned and typed schema that codifies the CDM conceptual model. The CDM implementation and its version history are provided in [3]. The schema is specified using the Apache Avro Interface Description Language (IDL) language [13]. Avro IDL is a high-level language for authoring Avro schemata. It provides a familiar feel for developers in that it is similar to common programming languages like Java, C++, and Python. The main advantage of using Avro is it decouples the schema from the serialized data: the reader schema which should be available to the reader at deserialization time, can be different than the writer schema. This means the serialized data on-the-wire is smaller in size since it does not need to have the typing information. More importantly, schemas can evolve over time independent of the data and without code generation.

Each serialized TCCDMDatum record can be any of the following: Host, Principal, ProvenanceTagNode, UnknownProvenanceNode, Subject, FileObject, IpcObject, RegistryKeyObject, PacketSocketObject, NetFlowObject, MemoryObject, SrcSinkObject, Event, UnitDependency, TimeMarker, EndMarker. Each record contains its type, a unique identifier of the host publishing this record, a sequence number that monotonically increases each time the system is started, and the instrumentation source that generated the record (e.g., Android Java or native, FreeBSD OpenBSM or dtrace, Linux auditd or netfilter, Windows ETW, etc.).

To avoid deserialization ambiguities, which specifically cause problems with the Python bindings, we make sure that



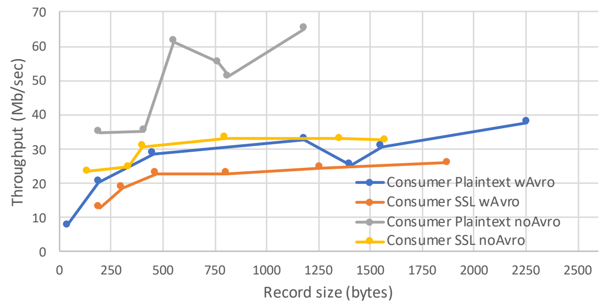


Figure 3: Throughput of the Java producer (top) and consumer (bottom) processes, producing and consuming CDM records. Plaintext means no TLS encryption/decryption of the Kafka messages, while wAvro means with Avro serialization/deserialization of the records at the producer/consumer.

either each record has a unique set of field names, or producers using the CDM use at least some of the unique optional fields when creating instances of records. This way serialized records of different types don't look the same on the wire.

Performance Results We evaluate the performance of the CDM mainly in terms of throughput of producing records by the tagging and tracking systems, and throughput of consuming them by the analysis technologies (Figure 1), and the end-to-end latency. Our goal from this evaluation was to ensure the throughputs supported by the infrastructure far exceed the rates at which the tagging and tracking technologies (analysis technologies) can produce (consume) data, and to ensure that CDM serialization/deserialization is not the bottleneck. Throughout the analysis, we use Apache Kafka [14] as the data handling substrate.

Figure 3 shows both the Java producer and consumer throughputs using CDM v20, Apache Avro 1.8.2, and a Kafka 1.1.0 cluster comprised of 3 brokers running with a single Kafka topic (with one partition replicated across two node). We plot the producer (consumer) throughput of serializing (deserializing) plaintext with and without Avro, and with and without TLS encryption. The producer throughput does not include the record creation time, which has negligible effect. In summary, we can sustain more than 10 MB/sec of throughput

Platform	Data Rate	Rec. Size	Rec. Rate
CADETS [12]	149 KB/s	255 B	600 rec/s
ClearScope [5]	33 KB/s	546 B	62 rec/s
FiveDirections	403 KB/s	216 B	1912 rec/s
THEIA [6]	185 KB/s	203 B	933 rec/s
TRACE [9]	291 KB/s	139 B	1406 rec/s

Table 1: Observed mean data rates, and mean record sizes, and calculated mean record rates by platform for the different tagging systems. Record rates calculated by dividing mean data rate by mean record size. Data rates are captured over a two hour window during the course of an evaluation event.

for the desired record sizes of the tagging and tracking technologies which range from 33 KB/sec to 400 KB/sec as shown in Table 1. This is orders of magnitude more than the anticipated record rates. For the producer, TLS encryption has a significant effect on throughput while for the consumer, Avro deserialization has a significant effect. Finally, end-to-end latency was on the order of 2 milliseconds, which is negligible relative to the timescales of the detection algorithms.

4 Conclusions

This paper presented a common data model, a rationale for its design, and an open source implementation. The unified data model enabled several real-time detection and forensic analysis technologies to consume the provenance events produced by a large breadth of tagging and tracking technologies operating at different granularities and on different platforms. Unlike existing provenance standard representations, such as W3C PROV, the common data model treats events and their complex semantics as first class citizens of the model, and incorporates granular provenance tags for tracking data flow with increased precision. We showed how a single producer process can stream serialized CDM records at more than 10 MB/sec over a Kafka queue, which exceeded the real time data rates at which tagging and tracking technologies streamed data. By its very nature of being a common model that needed to accommodate several tagging and tracking technologies, the model is over-specified. There are several opportunities to tune and reduce this model for a specific tagging and tracking technology to achieve the best efficiency, and to reduce semantic mismatch.

Acknowledgments

The CDM would not have been possible without the help of all TC performers. We especially thank R. Sekar, Vinod Venkatakrishnan, Scott Stoller, Ashish Gehani, Gabriela Ciocarlie, David Archer, Jeff Perkins, Michael Gordon, Yingbo Song, Arun Thomas, and Amanda Strnad for their insights and guidance designing the CDM.

References

[1] Transparent computing engagement 5 data release. https://drive.google.com/drive/folders/1okt4AYElyBohW4XiOBqmsvjwXsnUjLVf, 2019.

[2] Meisam Navaki Arefi, Geoffrey Alexander, Hooman Rokham, Aokun Chen, Michalis Faloutsos, Xuetao Wei, Daniela Seabra Oliveira, and Jedidiah R Crandall. Faros: illuminating in-memory injection attacks via provenance-based whole-system dynamic information flow tracking. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 231–242. IEEE, 2018.

[3] BBN. CDM API repository. https://github.com/raytheonbbn/transparent-computing, 2020.

[4] Yolanda Gil and Simon Miles. Prov model primer. http://www.w3.org/TR/2013/NOTE-prov-primer-20130430, April 2013.

[5] Michael Gordon, Jordan Eikenberry, Anthony Eden, Jeff Perkins, and Martin Rinard. Precise and comprehensive provenance tracking for android devices. Technical report, MIT, 2019. https://dspace.mit.edu/handle/1721.1/122968.

[6] Yang Ji, Sangho Lee, Mattia Fazzini, Joey Allen, Evan Downing, Taesoo Kim, Alessandro Orso, and Wenke Lee. Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC18, pages 1705–1722, USA, 2018. USENIX Association.

[7] Angelos Keromytis. Transparent Computing. DARPA Broad Agency Announcement, DARPA-BAA-15-15, 12 2014. https://www.darpa.mil/program/transparent-computing.

[8] Angelos Keromytis. Transparent computing engagement 3 data release. https://github.com/darpa-i2o/Transparent-Computing, 2018.

[9] Shiqing Ma, Juan Zhai, Yonghwi Kwon, Kyu Hyung Lee, Xiangyu Zhang, Gabriela Ciocarlie, Ashish Gehani, Vinod Yegneswaran, Dongyan Xu, and Somesh Jha. Kernel-supported cost-effective audit logging for causality tracking. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 241–254, Boston, MA, July 2018. USENIX Association.

[10] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, R Sekar, and VN Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1137–1152. IEEE, 2019.

[11] Md Amran Siddiqui, Alan Fern, Ryan Wright, Alec Theriault, David Archer, and William Maxwell. Detecting cyberattack entities from audit data via multi-view anomaly detection with feedback. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] Amanda Strnad, Quy Messiter, Robert Watson, Lucian Carata, Jonathan Anderson, and Brian Kidney. Casual, adaptive, distributed, and efficient tracing system (cadets). Technical report, BAE Systems Burlington United States, 2019.

[13] The Apache Software Foundation. Apache avro. https://avro.apache.org.

[14] The Apache Software Foundation. Apache kafka. https://kafka.apache.org.