A Common Data Model for Tagged Event Streams: Version 18

I. Introduction

This document is intended to describe the semantics and syntax of a common data model that will be used for capturing tagged event streams produced by TA1 performers. The core model is directly based on the proposals put forth by the three TA2 teams, which we refer to as MARPLE, ADAPT, and RIPE teams/proposals and evolved to include TA1 team concerns. The TC performers reached consensus on the core conceptual model described in this document. Our original attempt was to combine the strengths of all three proposals by borrowing key concepts from each, but the design evolved over time as the teams gained experience with the data model.

There are three main requirements articulated by the TA2 teams from such a data model. The model is expected to:

- be able to capture metadata about both the data and control planes with high fidelity and various granularities
- be natural for expressing the information produced by TA1 performers without adding undue burden on them
- contain the semantics needed to enable real-time detection and forensic analysis by TA2 performers

In § II, we discuss some of the high level concepts. We then present a work in progress typed schema specification of the conceptual model in § III which will be used directly for expressing and serializing TA1 data streams. This a graph data model so the entities and relationships correspond directly to nodes and edges in the graph. The history of changes to the data model is tracked in § IV

II. Conceptual View

There are six core entities in the model: *Hosts*, *Principals*, *Subjects*, *Events*, *Objects*, and *Tags*. The model combines the following strengths from the three proposals:

- MARPLE: MARPLE's event-based model served as the basis of the common data model. Events are first-class entities that are represented with high fidelity, along with both control flow (order) and data flow (tags). More on the rationale in § II-C.
- ADAPT: the hierarchical relationships proposed by ADAPT in PROV-TC for objects and subjects enable us to capture the mixed granularities and containment semantics. The Entity Relationship Diagram of PROV-TC was a useful tool for communicating the model.
- RIPE: data flows between objects and events are explicit in the model.

As important as the model itself is the rationale for making some of the different design choices which we describe in this section.

A. 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.

B. Subjects

Subjects 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 (while we can represent this granularity, as of now it is unclear if and when this will be useful).

C. Events

Events represent actions executed on behalf of subjects. Events could include system calls, 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 betwen subject and object or vice-versa. Events are assumed to be atomic so there is no direct relationship between events (except for the meta-event described shortly). Instead, events are related to other events through the affected subjects and objects.

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*.

D. Objects

Objects, 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.

a) 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 §II-E) 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 for the following reasons: values are one time constructs that only show up with the event and are not referenced afterwards during execution. TA-1 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.

b) Granularity and types: We discussed representing objects at various granularities. ADAPT modeled object containment using the *isPartOf* relationship associated with the object/artifact. This allows us to represent a set of objects being part of a parent object, such as when writing buffers (objects with different tags) to a file (parent object). To explicitly differentiate between different types of objects, we also decided to subclass the abstract object into Files, Network Flows, Memory, Packet Sockets, Registry Key, and Unnamed Pipe objects (and others we may idnetify) 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 RegistryKeyObject 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 (as proposed in ADAPT). 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. For now, 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).

c) Provenance: As discussed earlier, we explicitly model the relations between events and objects and subjects using the events as relationships (or edges) that connect subjects and objects. These edges represent provenance directly in the graph. The complementary more granular approach uses tags, discussed next.

E. Tags

The tag design was originally fleshed out by the MARPLE team, and then later reworked signficantly by the Clearscope TA1 team. The rationale for tags and their semantics is described next.

The primary motivation for tags is the ability to capture flows at a finer granularity and with increased precision. Specifically, tags can capture:

- *Increased precision:* Tags should be able to capture 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 TA1 system can indicate that it depends only on 10th and 97th input files.
- *Nature of dependence:* The behavior of the system can be understood in terms of a series of operations on inputs to produce outputs. Fine-grained tracking can identify the effect of these operations on data flows. These effects can be described using *tag operations,* which can capture:
  - common operations on data such as concatenation, mixing, compression, and so on.
  - *declassification* and *endorsement,* the central operations used in information flow control literature
  - strength of dependence, e.g., control dependence or implicit dependence

- *Flows involving internal (unreported) objects:* Especially for in-memory objects, not all updates and flows may be reported by a TA1 system. A TA1 system will internally keep track of these updates, and can then make the information available at the time when these objects contribute to an event argument.

Tags are directly associated with event parameters i.e., with Values, as attributes of the Value entity for fine grained taint tracking. Each Value has a array of tags associated with the bytes that make up the Value.

III. Typed Schema Definition

TA3 is developing a versioned and typed schema that codifies the above conceptual model. The actual data model will necessarily evolve based on the TA1-TA2 discussions expected in the coming months. The schema definition will evolve in parallel to match the latest version of the data model.

The schema is specified using the Apache Avro Interface Description Language (IDL) language. 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. Also, it is similar to IDLs in other serialization frameworks such as Thrift and Protocol Buffers. Avro provides a tool to convert the Avro IDL specification into an Avro schema in JSON format, which is used by Avro serialization frameworks for automatic serialization and deserialization of objects that conform to the schema. Avro (and the TA3 APIs) additionally provide the tools for automatic syntax checking to verify the data conforms to the typed schema.

Both the schema and this documentation are kept in the same git repository and will evolve together. The latest development snapshot containing the evolving schema is in a git repository shared with the rest of the TC team at the following URL https://git.tc.bbn.com/bbn/ta3-serialization-schema/tree/master/avro. Look for the latest version xx of the schema file *CDMxx.avdl*.

IV. History

Version	Changes
0.3	- Added File and Flow subclasses to Object - Added Value object - Added source to all entities - Marked that tags discussion is still in progress - Added location/size to the affects relation from obj to event
0.4	- Revamp of the Events - Revamp of the Objects - Revamp of the Tags - Incorporated material from tags writeup by Sekar/Venkat - Placeholder for tag structure pending discussion with Venkat
0.5	- Added description of some key attributes - Added schema section showing typed schema - Added memory subclass, disambiguate Value as its own entity - Moved location and size as optional attributes of event - Made sequence an attribute of event - Updated provenance tag description (structure is work in progress)
0.6	- Added SensorObject and SensorTypes to accommodate Android - Renamed Source to InstrumentationSource and defined a set of them - Added program point to event, and added blind event type - Added principal, does not yet model authentication - Added unique ids (uids) to all entities, important for streaming efficiency and for modeling edges - Refactored Tags, now ProvenanceTag entity has tagExpression, and conf/integ tags as attributes - Refactored timestamps, now long (we will add logical types when that is supported in IDL) - Refactored Object inheritance for AbstractObject, File, Netflow, Memory, Sensor - Added optional key-value pairs to all entities and edges for extensibility - Added start and end timestamps to subject and unitid to model unit instrumentation - Specified optional vs mandatory fields - Added meta event concept using hasParent edge for event
0.7	- Refactored Value; it is not a first class entity anymore, we made it an attribute of the event - Tag expression modeled as a tree using ProvenanceTagNode - Removed all edges previously associated with Value - Refactored tags representation: added ProvenanceTagNode that uses tree representation - Associated values with tags allowing run length encoding - Added SOURCE_WINDOWS_DIFT_FAROS to instrumentation sources - Added attributes to Subject for env vars: imported/exported libs, process information - Renamed SensorObject to SourceObject, representing generic source (sensor or other) - Made subject pid, ppid mandatory

0.8	- Rename entity uid attribute back to uuid to avoid confusion with user id - Rename SourceObject to SrcSinkObject: every sink can be a source but the opposite is not true - Rename SourceType to SrcSinkType - Add SourceType.SOURCE_SINK_IPC, MIT/ClearScope is using this to represent net flow and internal IPC - Add SourceType.SOURCE_SYSTEM_PROPERTY for representing program property variables - Added SOURCE_FREEBSD_DTRACE_CADETS, SOURCE_FREEBSD_TESLA_CADETS for CADETS instrumentation sources - Add EDGE_FILE_AFFECTS_EVENT, EDGE_NETFLOW_AFFECT_EVENT, EDGE_MEMORY_AFFECTS_EVENT, - Remove ProvenanceTagNode.numChildren, gets confusing with tag nodes referencing tagIds - Add ProvenanceTagNode attribute tag to the objects which subsumes Integrity and Confidentiality tags and is more expressive - Add optional k/v pairs to the ProvenanceTagNode for flexibility, using this in ClearScope for program point keys - Updated ProvenanceTagNode.value union, replaced string with long. This is to reference uuid for source/sink object - Add SOURCE_AUDIO to the SrcSinkTypes - Add optional attribute Event.name, common across datasets - Added unknown even types EVENT_KERNEL_UNKNOWN, EVENT_APP_UNKNOWN etc which will be used sparingly - Added edge types EDGE_EVENT_AFFECTS_SRCSINK, EDGE_SRCSINK_AFFECTS_EVENT for generic source/sink objects - Added Value.valueDataType to describe the type of the Event.parameters's value (not always byte[]) - Updated Value.tag run length encoding - Added Event.threadId since adding a separate thread entity seems overkill when its not adding any info
0.9	- Added InstrumentationSource SOURCE_LINUX_BEEP_TRACE - Added EventType EVENT_CLONE, EVENT_RENAME, EVENT_UNIT, EVENT_UPDATE - Added boolean isPipe field to FileObject - Added 256 bit UUID for all entities instead of the long (long term solution) - Added default value 0 for Event.sequence and made Event.timestampMicros optional - Needed for synthetic events where time or order is not known (e.g. some TRACE events) - Made MemoryObject.pageNumber optional - Fix typo in SrcSinkType.SOURCE_HEART_RATE
10	- Add SubjectType.BASIC_BLOCK - Make Subject.startTimeMicros optional (for processes that have started before monitoring starts) - Add ValueType enum and Value.type to distinguish input vs return values - Make Value more expressive to allow complex data types i.e. add Value.components=array<Value> - Add event types EVENT_RECVFROM, EVENT_RECVMSG - Add InstrumentationSource.SOURCE_LINUX_THEIA - Create a new entity TagEntity with a uuid and a ProvenanceTagNode (and perhaps a InstrumentationSource) that we can use for Subjects and Objects and Events - Add the EdgeTypes for subject/object/event tags and add an edge between events EDGE_EVENT_CAUSED_EVENT, EDGE_FILE_HAS_TAG, EDGE_NETFLOW_HAS_TAG, EDGE_MEMORY_HAS_TAG, EDGE_SRCSINK_HAS_TAG, EDGE_SUBJECT_HAS_TAG, EDGE_EVENT_HAS_TAG - Remove the AbstractObject.tag (replaced with the TagEntity and hasTag edge)
11	- Added enum ValueDataType for typing primitive values and complex values - Updated Value.valueDataType to be of type ValueDataType for stronger typing - Value.valueBytes should assume UTF_32BE encoding for all characters and strings - Updated docs for Value.tag, Value.size, and Value.valueBytes with examples - Added TCCDMDatum.CDMVersion=11 field to keep the CDM version explicit in the data
12	- Add VALUE_DATA_TYPE_BOOL data type: a boolean will be represented as a single byte TRUE=1, FALSE=0 - Value.valueDataType is now mandatory, used to be optional - Added flag Value.isNull = true for null values - Added Value.runtimeDataType which is a string representing the runtime data type of the value - Updated documentation for Value record
13	- Add event types: EVENT_SENDTO send through socket, EVENT_SENDMSG send through socket, EVENT_SHM share memory between processes - Add timestamp to TagEntity since multiple tags may be associated with the same entity over time - This is to avoid having to rely on the edge timestamp - Changes to support five directions data (windows): (1) Add RegistryKey object, (2) Add Value.name string: events that read/write from registry can specify the registry values as Values (3) Add Netflow protocol attribute, optional (TCP=6, UDP=17, IPv4=6, etc) (4) Add the edges EDGE_REGISTRYKEY_AFFECTS_EVENT, EDGE_REGISTRYKEY_HAS_TAG, EDGE_EVENT_AFFECTS_REGISTRYKEY

14	- Modify ProvenanceTagNode based on feedback from RIPE and ClearScope: (1) remove union value and tree structure (2) removed arbitrary key value pairs (3) increased tagId from int to UUID (4) added previous tag field (5) added opNode field - Add ProvenanceOpNode to combine tags - Incoming and outging edges to/from events are now embedded in the event record: (1) add subject (incoming) UUID and object/subject (outgoing) UUID fields to event record (2) remove event affects, affects event, and event is generated by edge types - Remove *_HAS_TAG edge types with no replacements - Remove EDGE_EVENT_CAUSES_EVENT edge type with no replacement - Add parent subject and local principal fields to subject record - Removes has parent and has local principal edge types - Remove simple edge record - Add optional fields âĂŸiterationâĂŹ and âĂŸcountâĂŹ to âĂŸSubjectâĂŹ record - Needed for distinguishing individual âĂIJunitsâĂİ of execution - Add optional âĂŸepochâĂŹ field to âĂŸAbstractObjectâĂŹ record - Needed to model when an object is deleted and a new one is created with the same identifier (such as a file path) - Add optional âĂŸsizeâĂŹ field to âĂŸMemoryObjectâĂŹ record - Needed to model the span of memory reported in mmap() / mprotect() calls - Remove notion of object versions, because TA2s will assign versions based on event sequence: (1) Remove EDGE_OBJECT_PREV_VERSION edge type (2) Remove âĂŸversionâĂŹ from âĂŸFileObjectâĂŹ and âĂŸRegistryObjectâĂŹ records (3) Remove lastTimestampNanos from AbstractObject record - Remove unknown source to force people to identify them - Change unknown event type to other event type to more accurately reflect what is being modeled - It's not that the reporter doesn't know the type, but that we haven't enumerated ALL types - Add FileObject types - Remove generic event pInfo field - Fields that TA1s need should be first-class attributes, possibly optional - Remove TagEntity record - Tags should always included as attributes of records, and not associated later - Increase timestamp precision to nanoseconds and make timestamps mandatory - Add TimeMarker record - Add more data Android data sources - Change event param value types to src, sink and control - Add UnnamedPipeObject - Add new event types - Add CryptographHash record and cryptographic hash types - Move Windows Program Execution (PE) info from Subject to FileObject - Add value to RegistryKeyObject - Remove file path/url from FileObject; instead include it in event records - This solves the issue of hard links where multiple paths reference a single file UUID. - Add optional Windows privilege level attribute to Subjects - Change src/dst IP address and port to inbound/outboud to make the semantics independent of the event (eg, connect vs accept) - Remove LocalAuthType since it is not currently used - Add username (or human-readable string identifier) to Principal

15	- Make Subject's parentSubject attribute optional and default value to null
	  - Identifies to consumer which parents have not been set, which is valid in cases where the publisher doesn't know the parent.
	- Value's size attribute is now -1 for non-array types (also: -1 default size)
	- Move source attribute to top level TCCDMDatum record
	  - Less useful and redundant in other records
	- Change Event's predicateObject to optional, because it's not always applicable
	- Add flowObject, subject, and systemCall attributes to ProvenanceTagNode
	- Reinstate memoryAddress in MemoryObjects, and allow optional decomposed components pageNumber & pageOffset
	- Remove UnnamedPipeObject's localPrincipal field
	  - This object type is transient, and the principal is already in the provenance chain as the controller of the subject that created the unnamed pipe object
	- Add optional fileDescriptor field to NetFlow and SrcSinkObjects
	- Add FileObjectType field to FileObject record
	  - This field was mistakenly missing in previous version
	- Change ProvenanceTagNode's flowObject to optional
	- Change TimeMarker's timestamp field's name
	  - This is a hack to overcome a bug with Python deserialiation of union records.
	- Change NetFlow's inbound/outbound to local/remote
	  - more intuitive names
	- Add UnitDependency edge record

16	- Add SOURCE_SINK_UNKNOWN SrcSinkType - useful when objects are opened before TA1 technology is started - Add EVENT_MODIFY_PROCESS for events that modify the process environment (eg, umask, chdir)
17	- Added new Android service types to SrcSinkType enum - Modified SrcSinkType's enum value prefix
18	- Adds CDM version number to namespace - Adds StartMarker and EndMarker records to delineate system/data stream start/stop - Adds PacketSocketObject - Adds EVENT_ADD_OBJECT_ATTRIBUTE, which is used to add attributes to an object that was incomplete at the time of publish. - Adds EVENT_FLOWS_TO, EVENT_UMOUNT, EVENT_SERVICEINSTALL - Replaces SOURCE_LINUX_AUDIT_TRACE in enum InstrumentationSource with SOURCE_LINUX_SYSCALL_TRACE and SOURCE_LINUX_NETFILTER_TRACE - TRACE is now using Audit's net filter records in addition to system call records. - Adds link, block special, char special file types - Adds 32-bit and 64-bit pointer value types - Adds optional source & sink UUIDs to UnnamedPipeObject. Also, makes source/sink file descriptor attributes optional - Adds Host record and associated records & enums to support cross-host provenance - Adds support for ProvenanceAssertion records - Changes Event's sequence attribute to optional for inferred events - Sets the @order("ignore") property for maps, so that they can be compared - Adds SOURCE_WINDOWS_MARPLE to InstrumentationSource enum