TAHOE – A Cyberthreat Language

Farhan Sadique

July 13, 2021

Contents

1 TAHOE — A Cyberthreat Language

Any CIS platform like CYBEX-P potentially handles hundreds of different data formats. Thus, it needs a standard data format and structure to represent threat data. A cyberthreat language (CTL) is a specification of how to format and serialize any kind of threat data. CYBEX-P uses TAHOE instead of other CTLs like STIX[2] or MISP[4]. TAHOE structures threat data as JSON [3] documents. In this section we introduce TAHOE as a better alternative to other CTLs and directly compare TAHOE with STIX and MISP. The complete TAHOE specification is available on GitHub [5].

1.1 TAHOE Data Instance

A piece of TAHOE data is called an `instance` and there are 5 types of TAHOE `instances` —

1. **Raw** A `raw` data instance stores unprocessed user data.
2. **Attribute** The most basic datatype that holds a single piece of information, like an IP address. Fig. 1a shows an email address `attribute`.
3. **Object** Groups several `attributes` together, e.g., a file `object` may have a filename and a file-size `attribute`. Fig. 1f shows a TAHOE `object` with two `attributes`.
4. **Event** An `event` consists of one or more `attributes` or `objects` along with a `timestamp`. `Events` structure `attributes` or `objects` into complete threat data. Fig. 1h shows and email `event`.
5. **Session** A `session` groups arbitrarily related `events` (e.g. events when a user visits a website).


```
{"itype": "attribute",
"sub_type": "email_addr",
"data": "jdoe@example.com",
"_hash": "5f07..."}
```

(a) Attribute 'from-email-addr'

```
{"itype": "attribute",
"sub_type": "name",
"data": "John Doe",
"_hash": "22af..."}
```

(b) Attribute 'from-name'

```
{"itype": "attribute",
"sub_type": "email_addr",
"data": "mary@example.com",
"_hash": "b591..."}
```

(c) Attribute 'to-email-addr'.

```
{"itype": "attribute",
"sub_type": "name",
"data": "Mary Smith",
"_hash": "4c90..."}
```

(d) Attribute 'to-name'.

```
{"itype": "attribute",
"sub_type": "subject",
"data": "Saying Hello",
"_hash": "50f2..."}
```

(e) Attribute 'email-subject'.

```
{"itype": "object",
"sub_type": "from",
"_cref": ["5f07..","22af.."],
"_ref": ["5f07..","22af.."],
"_hash": "d722..."}
```

(f) Object 'from'.

```
{"itype": "object",
"sub_type": "to",
"_cref": ["4c90..","b591.."],
"_ref": ["4c90..","b591.."],
"_hash": "da09..."}
```

(g) Object 'to'.

```
{"itype": "event",
"sub_type": "email",
"orgid": "test_org",
"timestamp": 880127706.0
"_cref": ["50f2..","da09..","d722.."],
"_ref": ["4c90..","d722..","5f07..",
    "22af..","da09..","b591..","50f2.."],
"_hash": "f70b..."}
```

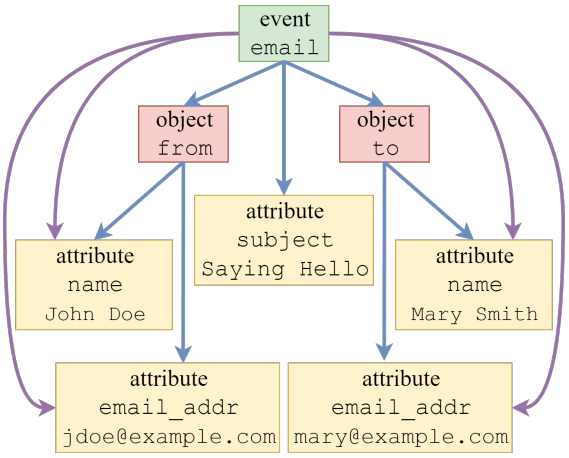
(h) Event 'email'.

Fig. 1: An email event in TAHOE Format. The complete representation of the email consists of 8 JSON documents. Each document is a node in the TAHOE graph. `_hash` is the globally reproducible and unique ID of a document. `_cref` stores the edges from objects and events to their children. `_ref` stores the edges to both children and grandchildren and so on. The graph is visualized in Fig. 2a.

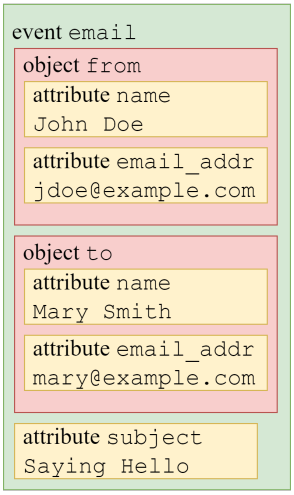
1.2 Data Structured as Graphs

TAHOE structures data as graphs, where each TAHOE instance is a graph node. Fig. 1 shows an email structured in TAHOE format. Fig. 2a visualizes the email as a TAHOE graph with 8 nodes and 11 edges.

As seen from Fig. 1, attributes store actual value or data. Objects and events, on the other hand, do not store any actual data. So, the complete representation of the object in Fig. 1f must include the attributes



(a) The email event as a graph with 8 nodes and 11 edges.



(b) The email event as a nested document.

```
"email": {
  "from": [{
     "email_addr": [
        "jdoe@example.com"],
     "name": ["John Doe"]
     }],
  "to": [{
     "email_addr": [
        "mary@example.com"],
     "name": ["Mary Smith"]
     }],
  "subject": [
     "Saying Hello"]
}
```

(c) The email event as a nested JSON document.

Fig. 2: The email event (green) from Fig. 1 as a TAHOE graph and a nested document of objects (red) and attributes (yellow).

in figures 1a and 1b. Similarly, the complete representation of the email event in Fig. 1h must also include the other 7 documents.

As shown in Fig. 1 all TAHOE documents have a field called `"_hash"`. The value of this field is derived from the SHA256 hash of the document and serves as a unique identifier for this document. For example, the string '`attributesubject"Saying Hello"`' is an unique representation of the subject attribute in Fig. 1e. We can calculate the SHA256 checksum of this string to be '`50f2...`'. This checksum is the unique identifier of the attribute `subject="Saying Hello"` in any TAHOE database.

Ojects and events have an array field called `"_cref"`. The `"_cref"` field stores graph edges. For example, the `"object-from"` in Fig. 1f has `"_cref"= ["5f07..","22af.."]`. This indicates, the object is connected to the attributes in Fig. 1a and in Fig. 1b. Similarly, the event in Fig. 1h is connected to two objects in Figures 1f and 1g and also the attribute in Fig. 1e. Fig. 2a shows these edges as blue arrows.

The array `"_ref"` stores graph edges to all subsequent nodes including children and grandchildren. This field is used for making queries faster and explained in subsection 1.4. Both the blue and purple edges from Fig. 2a are stored in `"_ref"`. Note, how the event contains complete information about the email just by referring to the other objects and attributes.

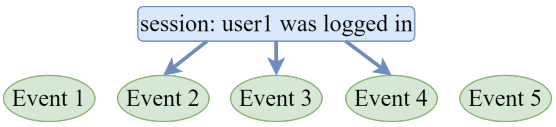


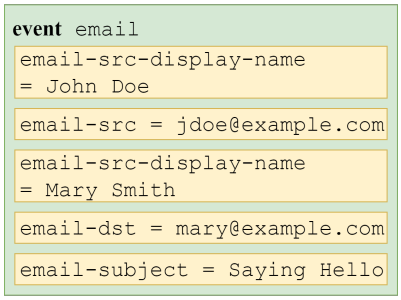
Fig. 3: Events grouped by arbitrary session parameter.

Finally, A TAHOE session is an arbitrary grouping of related events. This allows us to group events based on any condition the user desires. The session in Fig. 3 groups 3 events, recorded while *user*1 was logged in.

Events Viewed as Nested Documents Although, TAHOE structures events as graphs, they can be viewed as nested documents. Fig. 2b shows the email event from Fig. 1 as a nested document. Furthermore, Fig. 2c shows the JSON representation of the nested document. This JSON document is obtained by traversing the graph starting from the event node. The graph edges are stored in the `_cref` arrays. Analysts can choose to

view an event as a document or as a graph depending on their need. For all kinds of machine analysis (e.g query), however, the graphical structure of Fig. 2a is more suitable.

1.3 Representing Complex Data – TAHOE vs. MISP



(a) The email in Fig. 1 as MISP format. MISP struggles to structure complex data and specifies non-intuitive `attribute` types.

```
{
"0": {
    "type": "email-addr",
    "value": "jdoe@example.com",
    "display_name": "John Doe"
    },
"1": {
    "type": "email-addr",
    "value": "mary@example.com",
    "display_name": "Mary Smith"
    },
"2": {
    "type": "email-message",
    "from_ref": "0",
    "to_refs": ["1"],
    "date": "1997-11-21T15:55:06Z",
    "subject": "Saying Hello"
    }
}
```

(b) The email in Fig. 1 as STIX format.

```
{
"0": {
    "type": "ipv4-addr",
    "value": "1.2.3.4",
    "belongs_to_refs": ["3"]
    },
"1": {
    "type": "ipv4-addr",
    "value": "2.3.4.5"
    },
"2": {
    "type": "network-traffic",
    "src_ref": "0",
    "dst_ref": "1",
    }
"3": {
    "type": "as"
    "number": 42
    }
}
```

(c) A network packet in STIX format.

Fig. 4: The email event from figures 1 and 2 in MISP and STIX format and a network packet in STIX format.

Traditional CTLs like MISP often struggle to represent complex data. For example, Fig. 4a shows the email event from Fig. 1 in MISP [4] format.

The key problem is, `email-src` and `email-dst` are two different attribute-types. So, to fetch all emails to and from `jdoe@example.com`, one has to perform 2 queries – `email-src = jdoe@example.com` and `email-dst = jdoe@example.com`. Moreover, MISP represents a lot of information in the attribute-type. So MISP data structure includes cumbersome attribute-types like `passenger-name-record-locator-number` or non-intuitive attribute-types like `filename|md5`, `filename|sha224`, `filename|sha256` etc.

Firstly, TAHOE can store arbitrarily complex data because TAHOE objects can be infinitely nested and can refer other objects. As seen if Fig. 1, TAHOE has simple attribute types like, `email_addr` or `subject`. Secondly, a TAHOE event is connected to all of its attributes via the `_ref` array. So, to fetch all emails to and from `jdoe@example.com`, TAHOE requires only 1 query – `email_addr = jdoe@example.com`.

1.4 Indexing & Scalability – TAHOE vs. STIX

Earlier in section **??**, we claimed that, "While STIX is perfect for mutual data sharing, it is unscalable for any kind of data analysis." This subsection provides a detailed explanation in support of our claim. Since, both STIX and TAHOE use JSON documents we will have to store them in a NoSQL database. For this discussion, we consider the most popular NoSQL database - MongoDB [1].

Consider, the STIX document in Fig. 4b. It has 3 keys - `"0"`, `"1"`, `"2"`. Here, `"0"` and `"1` are JSON objects with 3 keys each whereas `"2"` is a JSON object with 5 keys. Assume, we want to *fetch all emails from jdoe@example.com*. The MongoDB syntax for that query is, `find({"0.value":"jdoe@example.com"})`.[1] This query will get all JSON documents in the database which have `"0.value"="jdoe@example.com"`. Similarly, if we want to *fetch all emails with subject="Saying Hello"* the query is `find({"2.subject": "Saying Hello"})`.

[1] The actual query is `find({"0.type":"email-addr","2.type":"email-message","0.value":"jdoe@example.com"})`. We have shortened the queries in the example for clarity.

Now, these queries will take forever in a decent sized database unless we index the keys `"0.value"` and `"2.subject"`. Similarly, in Fig. 4c, to efficiently lookup all the network traffic events of `"type"="as"`, we must index the `"3.type"` key. Eventually, for these two event types, we must index a total of 16 keys – `"0.type"`, `"0.value"`, `"0.display_name"`, `"1.type"`, `"1.value"`, `"1.display_name"`, `"2.type"`, `"2.from_ref"`, `"2.to_refs"`, `"2.date"`, `"2.subject"`, `"0.belongs_to_refs"`, `"2.src_ref"`, `"2.dst_ref"`, `"3.type"`, `"3.number"`. Indexing in this manner creates 3 problems for us:

1. Not all documents have all the keys. For instance, the network traffic event in Fig. 4c does not have the `"2.subject"` key. So, the indexing will be inefficient.
2. MongoDB only allows 64 keys to be indexed in a database collection. As discussed earlier, we have 16 keys only for two types of events. As we encounter, more event types with arbitrary structures, we will have hundreds of keys that need indexing. Indexing so many keys is not feasible.
3. Some, fields have large values. For example, RFC2322 states that the email subject has no length restrictions. Which means the `"2.subject"` field in Fig. 4b can be larger than 1024 bytes. However, MongoDB cannot index a field larger than 1024 bytes. So, large fields in a STIX document can never be indexed.

Going back to our earlier discussion, queries like `find({"0.value": "jdoe@example.com"})` will be too slow in a database if the fields are not indexed. That is why we claim that STIX is unscalable for data analytics.

TAHOE incorporates a novel solution to these challenges. Consider the query *fetch all emails with subject="Saying Hello"*. This is a two-step process in TAHOE –

1. We create the string '`attributesubject"Saying Hello"`'. This string is an unique representation of this subject attribute. We then calculate the SHA256 checksum of this string to be '`50f2...`'. This checksum is the unique identifier of the attribute `subject="Saying Hello"` in any TAHOE database.
2. In the second step, we perform the following MongoDB query - `find({"_ref": "50f2..."})`. [2] [3]

These two steps will return all the emails which have `subject="Saying Hello"`. In terms of TAHOE graph, we are essentially querying all events which are connected the TAHOE attribute `subject="Saying Hello"`.

Now, in TAHOE, we only query one key - `"_ref"`; so, we only index this one key. Therefore, we will never pass the 64 keys limit of MongoDB. Also, all events have the `"_ref"` field, so the indexing will be efficient. Finally, each element in the `"_ref"` array is a SHA256 checksum, which means each of them are 256 bits long. So, we will not violate the 1024 bytes limit on indexed fields. Thus TAHOE takes care of all the 3 problems mentioned earlier for STIX.

Fig. 5 compares the scalability STIX 2.0 and TAHOE. The graph compares the time take for the `find({"0.value": "jdoe@example.com"})` query for the STIX database with the `find({"_ref": "50f2..."})` query for the TAHOE database. The x-axis is the total number of email events in the database. The y-axis is the total time taken for 100 queries in the databases. The plot clearly shows that the time-required grows linearly with the database size for STIX. This is because the key `"0.value"` is not indexed and every query becomes a linear search in the database. The time-required for the TAHOE database, however, stays constant as the number of emails grows in the database. This is because the key `"_ref"` is indexed in the database. And since indexed keys are stored as hash-tables in the RAM, every lookup takes roughly the same time despite the size of the database.

1.5 Intrinsic Correlation of Graphical Data

Traditional CTLs, store threat data as separate documents like shown in Fig. 6a. These data are difficult to analyze because the events lack any direct correlation with own attributes. TAHOE, on the other hand,

[2] The actual query is `find({"itype":"event", "sub_type":"email", "_ref": "50f2..."})`.

[3] We have made a TAHOE backend library that interfaces with MongoDB to automate the whole query process. So, users will not actually have to calculate the checksums and then query the MongoDB. The library is available at https://github.com/cybex-p/tahoe

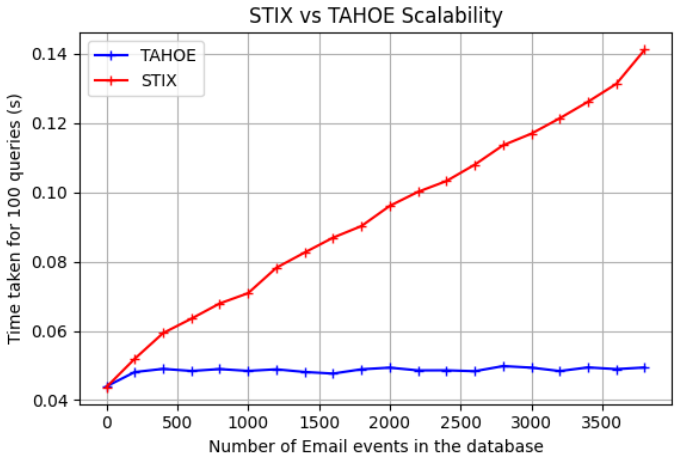
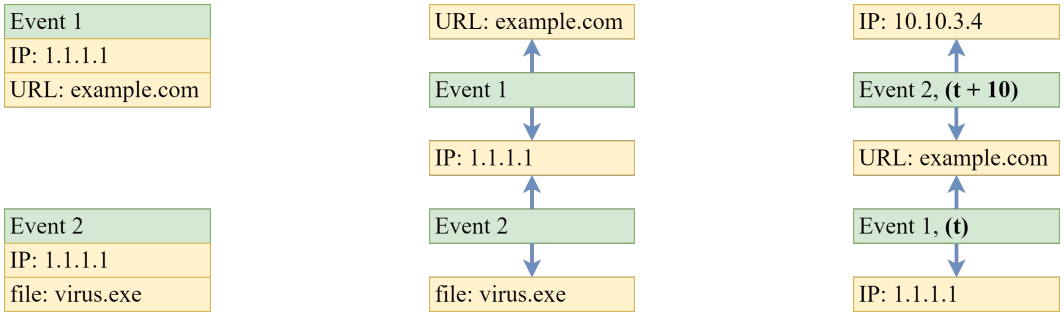


Fig. 5: Scalability of STIX vs TAHOE. The x-axis is the total number of email events in the database. The y-axis is the total time taken for 1000 queries in the databases. The query for the STIX database is `find({"0.value": "jdoe@example.com"})`. The query for the TAHOE database is `find({"_ref": "50f2..."})`.



(a) Two events as separate documents. (b) Two events as TAHOE graph. (c) A time-varying attribute.

Fig. 6: Fig. 6b shows how TAHOE intrinsically correlates the two separate events from Fig. 6a based on their common attribute. Fig. 6c shows an example TAHOE graph for a time-varying attribute.

represents data as graphs like in Fig. 6b. Here, two separate events are automatically connected by their common `attribute` (`1.1.1.1`) in TAHOE. Such 'intrinsic correlation' is a powerful feature of TAHOE, because if someone looks up `example.com` she will immediately see that `virus.exe` is related to it. This is a major strength of our investigation tool (subsubsection **??**). Moreover, we leverage this feature to formulate a novel malicious event detection mechanism in section 2.

Furthermore, TAHOE can correlate time-varying attributes. As shown in Fig. 6c, lets assume the Tahoe database records two events - Event 1 at time **t** and Event 2 at time **t + 10**. Also assume, During event 1, at time **t**, the domain `example.com` resolves to `1.1.1.1` and during event 2, at time **t + 10**, the domain `example.com` resolves to `10.10.3.4`. Now, the TAHOE database will connect both the IP addresses to the domain in a graph like that shown in Fig. 6c.

Please note that, both IP addresses are associated with `example.com` after Event 2 is recorded. Also note that, the IPs are not directly connected to `example.com`, rather connected via their respective events. For example, `1.1.1.1` is connected to `example.com` via Event 1 (time **t**) and `10.10.3.4` is connected to `example.com` via Event 2 (time **t + 10**).

So, if someone queries `example.com`, they will be able to see the complete graph in Fig. 6c including the timestamps. From, the graph any person or machine can deduce that `example.com` was associated with `1.1.1.1` at time **t** and was associated with `10.10.3.4` at time **t + 10**. Thus, the time aspect of the

relationship is preserved.

Moreover, users can specify time ranges when querying the TAHOE database to filter out one of these events. For example, if any user or machine queries events related to `example.com` between **t + 5** and **t + 15**, they will only see Event 2 in the graph.

1.6 Features of TAHOE

1.6.1 Data Normalization

TAHOE normalizes different formats of same type of data. Consider two firewalls from two different vendors. Their log data will be formatted differently despite having same type of data. TAHOE normalizes such differences by converting them into the same structure.

1.6.2 Data De-duplication

TAHOE prohibits duplicate data. For example, there can only be one instance of the IP 1.1.1.1 in a TAHOE database. This saves CYBEX-P a lot of storage by not storing the same IP in different `events`. TAHOE achieves this de-duplication of data by creating a globally reproducible hash of the data.

1.6.3 Database Independence

Although TAHOE is a graph based CTL we did not use a graph database as a container for TAHOE. In other words, all the information, including the edge data, of a TAHOE graph is stored in the JSON documents of the TAHOE `instances`, as shown in Fig. 1.

Furthermore, as described in 1.7 we have developed a universal threat data query language (TDQL) to communicate with any TAHOE storage. These two contributions make TAHOE a database-independent CTL.

1.6.4 Optimized for Indexing

Subsection 1.4 discusses how TAHOE is optimized for indexing in databases and compares the query performance of a STIX database with that of a TAHOE database. The ability to query related data is a novel feature of TAHOE and enables CYBEX-P to perform advanced analytics on TAHOE data.

1.6.5 Globally Unique & Reproducible Data for Conflict-free Sharing

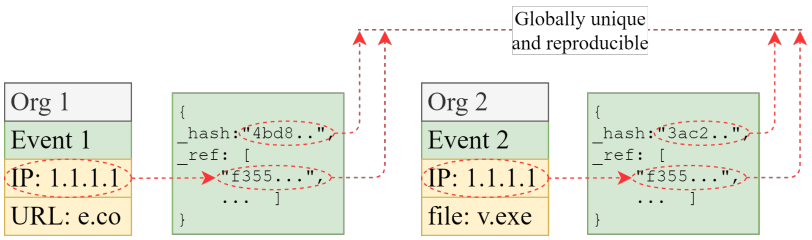


Fig. 7: TAHOE id and edges are globally unique and reproducible, making them collision free.

TAHOE data are globally unique and reproducible. As shown, in Fig. 7, the IP `1.1.1.1` has the same unique id (its hash) in two different organizations. Consider, `Org 1` shares `Event 1` with `Org 2`. If `Org 2` had a different id for `1.1.1.1` it would have to update the `_ref` array of `Event 1`. But, as hashes are reproducible yet unique, this is not required. Note that, event hashes include a timestamp (not shown in figure). Hence, two separate events will have different hashes even if they have same attributes.

1.6.6 Bidirectional Edges for Versatile Queries

TAHOE edges are bidirectional. As seen in Fig. 7, edge data is stored in the event only. This is because, an IP like 8.8.8.8 (public DNS) can potentially get connected to millions of events. If we store the hash of all these events in the IP attribute, it would result in an unbounded growth of its edge array. So, we store the edge info in the events. However, it takes only one pass over the database, to get all events that have a particular hash in their edge array. So, the edges are bidirectional for all intents and purposes.

1.7 Threat Data Query Language (TDQL)

TAHOE aims to standardize the structuring of threat data in terms of attributes, objects, events and sessions. This would allow users to query threat data using those terms. An example query could be fetch all events which include the attribute 1.1.1.1. At present this is not possible because event or attribute are not standardized terms for any existing database. For example, if a person queries an SQL database for events it would not know what to return, because event is not a standard term for SQL.

To that end, we have developed a universal threat data query language (TDQL) for TAHOE. TDQL acts as a layer between a database and a user. Additionally, TDQL is tailor made for threat data and addresses their nuances. While SQL depends on the structure of database tables, TDQL speaks in terms of attributes, objects, events etc. So, irrespective of the data storage or delivery protocol, a user can always fetch any threat data from any database.

Additionally, having a dedicated TDQL makes TAHOE, database-independent. However, detailed documentation of TDQL is beyond the scope of this research work.

2 ThreatRank to Detect Malicious Events

Earlier in subsection 1.5 we introduced how TAHOE intrinsically correlates data. Here, we extend upon it by formulating an algorithm, called ThreatRank, to assign a malicious score to each event in a TAHOE database. The score essentially sorts the events from most malicious to least malicious. In **??** we justify this algorithm with real data.

Consider, $\mathbb{A} = \{a_1, a_2, ..., a_m\}$ is the set of all attributes and $\mathbb{E} = e_1, e_2, ..., e_n$ is the set of all events. $\mathbb{E}_{mal} \subseteq \mathbb{E}$ is the set of known malicious events. We define $\mathbb{I}_{mal} = \{k \mid e_k \in \mathbb{E}_{mal}\}$. We want to determine the ThreatRank (TR) of a new event e_p.

We define $w_{i,j} = \{e_i, ..., a_x, e_y, a_z ..., e_p\}$ as the j^{th} path from e_i to e_p. Note that, the path encounters attributes and events in an alternating fashion and has distinct nodes.

Then the contribution of $w_{i,j}$ to the ThreatRank of e_p is calculated using the recurrence equation—

$$TR_{w_{i,j}}[k] = 0.998^{d_{k-1}} \times \frac{TR_{w_{i,j}}[k-1]}{L(w_{i,j}[k-1])} \tag{1}$$

where, $TR_{w_{i,j}}[1] = -1$; $d_k = 0$ for an attribute and for an event, d_k is the number of days passed since the event e_k was recorded; $L(x)$ is the degree of node x.

Assume, there are t_i paths from e_i to e_p. We define the set $\mathbb{W} = \{w_{i,j} \mid i \in \mathbb{I}_{mal}; j \in [1, t_i]\}$. $\mathbb{W}$ basically includes all the paths from all known malicious events to the new event. The total ThreatRank of e_p is then calculated as —

$$TR(e_p) = \sum_{w \in \mathbb{W}} TR_w[t_i] \tag{2}$$

Algorithm 1 lists the pseudocode for ThreatRank. The code is written using TAHOE terminology.

2.1 Why 0.998?

We multiply the ThreatRank of each event by 0.998^{d_k}. Here, d_k is the number of days passed since the event e_k was recorded. The value 0.998 is chosen such that after 1 year an event is half as significant

```
Input: 𝔼, 𝔼_mal
Function getRelated(node)
    if node.type = "event" then
        return node._ref
    end
    related ← []
    for event in 𝔼 do
        if node in event._ref then
            related.append(node)
        end
    end
    return related
end
Function findPaths(src, dest, currentPath)
    if src = dest then
        return currentPath
    end
    related ← getRelated(src)
    paths ← []
    for r in related do
        if r in currentPath then
            continue
        end
        paths.append(findPaths(r, dest, currentPath+[r]))
    end
    return paths
end
Function threatRankPath(path)
    tr ← −1
    for node in path do
        L ← degree(node)
        d ← 0
        if node.type = "event" then
            d ← node.daysOld
        end
        tr ← tr × 0.998**d / L
    end
    return tr
end
Function threatRank(newEvent)
    allPaths ← [],   TR ← 0
    for event in 𝔼_mal do
        allPaths.append(findPaths(event, newEvent, []))
    end
    for path in allPaths do
        TR ← TR + threatRankPath(path)
    end
    return TR
end
```

Algorithm 1: ThreatRank

($0.998^{365} = 0.48$) as a recent event ($0.998^{0} = 1$). The same event is only one-fourth as significant ($0.998^{730} = 0.23$) after two years. However, the user can choose a different value of this factor between 0.00001 and 1.0. A larger value indicates that older events are more significant in calculating the ThreatRank.

2.2 Who Classifies Malicious Events & Edges?

Malicious events or edges can be manually classified in three ways — (1) by CYBEX-P admin after analysis (2) by user voting (3) automatically for some data. For example, an IP that tries to connect to a honeypot, is automatically classified as a malicious IP in this context.

3 Data Governance & Privacy Preservation

CYBEX-P offers a robust data governance mechanism with granular access control of data. Here, we discuss the 'attribute based access control' protocol of CYBEX-P.

3.1 Data Model and Privacy Parameters

CYBEX-P converts any incoming data into a TAHOE event. Fig. 8 shows an event with two attributes – an IP and a file (filename). Assume, the data owner Org 2 wants to share the file with everyone but not the IP.

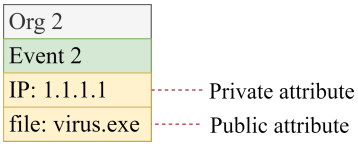


Fig. 8: Data owner determines if an attribute is public or private.

3.2 Public Attribute (Not Encrypted)

Data owner Org 2, first, converts the document into a TAHOE event as shown in Fig. 9. Here, 0xABC is the hash of the IP 1.1.1.1, 0xDEF is the hash of the file virus.exe, 0x123 is the hash of the event itself, and acts as the event id. The hashes of the attributes are placed in the edge array creating a graph. Note that, we use the term edge to denote the _ref array from 1.2.

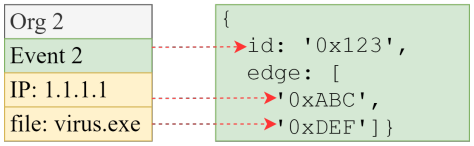


Fig. 9: Event data in TAHOE format before encryption.

Threat Model

The data owner Org 2 trusts all participants of CYBEX-P with the public attribute virus.exe.

Public Data Query

Now, assume a user wants to get all the events with the file virus.exe. She first generates the hash of virus.exe as 0xDEF. Then she looks up the database for events that have 0xDEF in the edge array. She will get the event 0x123 in return.

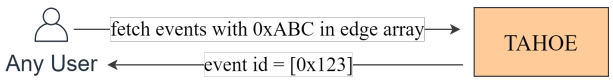


Fig. 10: Public Data Query.

3.3 Private Attribute Encryption

To protect the private attribute, 0rg 2 encrypts its hash 0xABC with secret to generate the ciphertext 0x789, as shown in Fig. 11. The owner can use any symmetric encryption technique of choice although TAHOE recommends AES256.

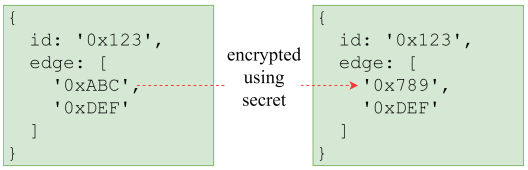


Fig. 11: Event data in TAHOE format after encryption.

Threat Model

The data owner 0rg 2 does not trust anybody including CYBEX-P with the private attribute 1.1.1.1.

Private Data Query

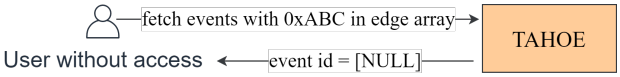


Fig. 12: Private Data Query.

Now, a user wants to fetch all the events with the IP 1.1.1.1. She first generates the hash of IP 1.1.1.1 as 0xABC. Then she queries the database for events that have 0xABC in their edge array. However, the database will return nothing, because the value 0xABC is not present in any event edge. 0rg 2 has essentially encrypted the graph edge.

3.4 Private Attribute Sharing

At this point, 0rg 2 wants to share the private attribute 1.1.1.1 with 0rg 3. To achieve this, 0rg 2 shares the secret with 0rg 3. CYBEX-P facilitates this sharing by providing a key management system (KMS) (11 in Fig. **??**).

Threat Model

0rg 2 trusts 0rg 3 and wants to share 1.1.1.1 with 0rg 3. However, 0rg 2 does not trust CYBEX-P or any other user. 0rg 2 shares the encryption secret with 0rg 3 using CYBEX-P KMS.

Private Data Query

Now, 0rg 3 wants to fetch all events with the IP 1.1.1.1. She first generates the hash of 1.1.1.1 as 0xABC. Then she encrypts the hash with secret to generate the ciphertext 0x789. Finally, she queries the database for events that have 0xABC or 0x789 in the edge array. The database will return the event 0x123 along with public events which include 1.1.1.1. Note that, this query still makes one pass over the database.

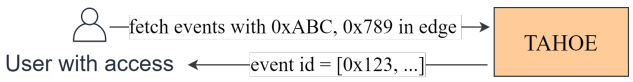


Fig. 13: Private Data Query by Trusted Party.

Private Data Correlation

A powerful feature of CYBEX-P is intrinsic correlation of data as described in 1.5. What makes TAHOE even more powerful is that, the intrinsic correlation mechanism works on encrypted data as well.

As explained in subsubsection 3.4, an authorized user can query encrypted data without revealing its value. The query performs a graph traversal, returning a complete graph of all the related `attributes` and `events`. This graph contains all the intrinsic correlations described in 1.5.

References

[1] Kyle Banker. 2011. *MongoDB in action*. Manning Publications Co.

[2] Sean Barnum. 2012. Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX). *MITRE Corporation* 11 (2012), 1–22.

[3] Tim Bray. 2014. The javascript object notation (json) data interchange format. (2014).

[4] Alexandre Dulaunoy and Andras Iklody. 2020. *MISP core format*. Internet-Draft draft-dulaunoy-misp-core-format-13. Internet Engineering Task Force. `https://datatracker.ietf.org/doc/html/draft-dulaunoy-misp-core-format-13` Work in Progress.

[5] Farhan Sadique. 2021. TAHOE. `https://github.com/cybex-p/tahoe`.