OCSF Schema Collaboration: Initial Decisions

Thank you for actively participating in the OpenSource CyberSecurity Schema Framework collaboration! While there has been some preliminary work completed to serve as a starting point, this document intends to highlight some pending decisions to be made that will affect the design and utilization of OCSF. We would appreciate consideration of the 3 proposals included here and would request feedback by April 23rd, either via e-mail response or in meetings to be scheduled throughout the week of April 18th.

Proposal 1: Schema Taxonomy, Categorization Design and Decision Flow (RFC/discussion)

Background:

The categorization of log records is a fundamental concept to aid in extracting value from log sources. The proper recognition of the content of a log helps with the identification of relevant data and aids in the process of aligning the data contained in the log with same or similar data from other logs, as well as offering a common path for normalizing the logs themselves. This is a fundamental consideration that will greatly affect interaction with normalized data.

A category is defined as a unique collection of classes, with each class being able to be in one and only one category. A class is a uniquely identifiable collection of OCSF objects and fields. Each category has a unique name and identifier, which can be utilized to categorize the records that fit the classes that make up the category.

Currently, there is a very rough utilization of categories, utilized uniquely in extensions. No categories yet been considered a core part of OCSF. This proposal is intended to be a net new starting point for core categories in OCSF.

The following request has been submitted:

> There is a need to ensure that categories are designed in a way to increase the value of the data and decrease any confusion or ambiguity for individual log source.
>
> OCSF should identify a clear path for matching a log source with the applicable OCSF class. OCSF categories should be relatively static as compared to classes. They should be broad enough to be utilized by a fair number of sources, and specific enough to provide value by normalizing those disparate logs.
>
> The following categories have been proposed:
>
> * Access Logs
> * Cloud Operations
> * Container Operations
> * Custom Application Logs
> * Database Activity
> * DNS Activity
> * Endpoint Activity / OS Logs
> * Identity
> * Management and Governance
> * Network Flow Activity
> * Security Alerts and Findings
> * Web Application Firewalls

Initial points of consideration:

- Can a log source provide logs to multiple classes and/or categories? (e.g., Windows Events Logs could populated a multiplicity of the categories listed below) Do we intend for a singular OCSF rendered schema to be applied to a log source or an individual log record?
- Should a log be categorized based upon content (disposition) or similarity (fitness)?
- How might a disposition_id/activity_id/outcome_id relate between a class and a category?
- As a thought exercise, auth logs may be considered. From an analyst POV, it might be considered “Identity” - “This user was authenticated by sshd/pam.d/etc.” However, as they are generally accessing an endpoint, an analyst might expect these logs to be characterized as Endpoint Activity/OS Logs. In a more esoteric argument, it could be said that the log is depicting “access”, so therefore it is actually an “Access” log. Finally, while not in the list above, a perhaps glaring omission is an “Audit” category.

Proposal 2: Remove ‘disposition_id’ in favor of ‘activity_id’ and ‘outcome_id’

Background

The disposition_id is a field that represents a log’s disposition as an enum. It is present in the base_event and is intended to be utilized as an optimized filter for certain events; e.g., for DNS activity, the enum might be:

```
-1 Other
0 Unknown
1 Resolved (The DNS request was successfully resolved)
2 Failed (The DNS request failed)
3 Query (The logs pertains to the DNS query itself)
```

The enum is unique to each OCSF class and must be documented.

The following proposal has been submitted

> Currently, Alert category classes use both disposition_id and activity_id attributes, and the the other classes, outside the Alert category, use disposition_id alone.
>
> There are a few issues with the current names and usage:
> • using a disposition_ id in the Activity classes is a somewhat misleading
> • searching for activities across all classes; for example to find all events that report a file was created, the query will have to use disposition_id and activity_id to check for a file create activity.
>
> **Activity Classes: Rename disposition id as activity id**
> Activity classes report behavior or an action of a particular kind, successful or not. Thus, using activity_id makes more sense than disposition id. In addition, use status id (Success/Failure) to indicate if the activity was either successful or if it failed. For example DNS Alert Event:
>
> activity_id 3 – Query: DNS query request
> status_id 1 – Success: DNS query request is successful
>
> The activity_id enum value names should be in a present tense. For example, Query or Delete.
>
> **Alert Classes: Rename disposition id as outcome id**
> Alert classes report outcome of actions that security products took when some activity occurs. For example DNS Alert Event:

> activity_id 3 – Query: DNS query request
> outcome_id 2 – Blocked: DNS query request was blocked
> status_id 1 – Success: Outcome is successful (nothing to worry about)
>
> The outcome_id enum value names should be in a past tense. For example, Blocked Or Allowed.

Initial points of consideration:

- Utilizing a versatile strategy between classes/categories may be premature, particularly as we do not yet have an agreed upon core set of schema categories or classes. Treating certain categories or classes to a different base_event set of fields and objects may introduce more volatility and complexity than necessary.
- Conversely, logs are not a holistic and uniform entity to be best summarized by a narrow enum. An additional enum for classification may be necessary for the concept of disposition_id to be beneficial and fully realized.
- Conceptually, there is a sustained concern about a growing complexity that will render the schema friendlier for computational scenarios over security personnel themselves. An increasingly complex schema may even discourage adoption and utilization. While it is unlikely that this change alone may cross that ambiguous threshold, attention should be paid to the overall effect on schema use cases and adoption.

Proposal 3: Profiles

Background

OCSF allows a schema to be defined by combining fields and objects into a unique set that is identified as a “class”. These classes may be grouped alongside similar classes and may be identified together through the use of “categories”. Classes and categories each have a unique identifier present in each record. The class ID is unique for each class, and the category ID is unique amongst categories, but common in the classes in the same category.

To support the industries use cases, the resultant classes generated via OCSF may be large enough in scale to make finding a relevant class challenging. The “fitness” of a certain class may also be poor, requiring either a dilution of strict requirements (making most of the fields/objects “optional”) or the generation of a new class (either to be accepted in the “core” or as an “extension”), which increases the work load on customers and diversifies the resultant normalization of logs.

The following proposal has been submitted

> **Profiles are overlays on event classes,** effectively a dynamic mix-in class of attributes and objects. While event classes specialize their category domain, a profile can augment event classes with a set of attributes independent of category. Multiple profiles can be added to an event class via an array of profile_uid values. This composition approach allows for reuse of event classes vs. creating new classes one by one that include the same attributes. Event classes and instances of events that support the profile can be filtered via the profile_uid across all categories.
>
> For example, a Malware profile that adds MITRE ATT&CK and Malware objects to system activity classes avoids having to recreate a new event class, or many classes, with all of the same attributes as the system activity classes. A query for events of the class will return all the events, with or without the security information, while a query for just the profile will return events across all event classes that support the security profile. A Host profile and a User profile can add Device, Process and User objects to network activity event classes when the network activity log source is a user’s computer.
>
> Three built-in profiles for Malware, Host and User are shown in the below table with their attributes.

	A	B	C
1	**Malware Profile**	**Host Profile**	**User Profile**
2	disposition_id / disposition		user
3	attacks	device	is_user_present
4	cvssv2	actor_process	user_entities
5	malware / other_malware	connection_info	accounts
6	quarantine_uid		user_result

> For example, if an OCSF class was defined (vastly simplified here) for a firewall, it might resemble the following:
> {
> “rule”: {...},
> “action”: {...}
> }
>
> While this might be sufficient for a network firewall, a firewall with a host profile might look like the following:
> {
> “rule”: {...},
> “action”: {...},
> “device”: {...},
> “actor_process”: {...},
> “connection_info”: {...}
> }
>
> A firewall with a user profile, which might be used on endpoint devices, might look like the following:
> {
> “rule”: {...},
> “action”: {...},
> “user”: {...},
> “is_user_present”: {...},
> “user_entities”: {...},
> “accounts”: {...}
> “user_result”: {...}
> }
>
> All of these example classes would be able to share the same class_id while presenting a different resultant schema due to the different profiles being applied.
>
> Other profiles can be product oriented, such as IDS, VA etc. if they need to add attributes to existing classes. They can also be more general, platform oriented, such as for Cloud or Windows environments.
>
> For example, AWS resources log events with an ARN (AWS Resource Name) and an AWS Account. An AWS specific profile can be added to any event class or category of classes that includes arn and IAM account attributes.
>
> Profiles can be used for enrichment of a specific product: e.g. Splunk Behavioral Analytics needs source and destination enrichment. A BA profile can add the enrichment to the relevant event classes.

Initial points of consideration:

- Although not specifically proposed for this use case, this would address a latent concern about implementation-specific artifacts which may not adhere to a strict reading of the schema. For example, partitioning data may be performed that would result in additional fields that are not defined by the class (e.g., year, month, day, hour). These fields are based upon the data (e.g., dividing a timestamp), but serve a functional purpose, and are not primarily intended to represent directly the data in the log record itself. While these fields could be included in the schema directly, as they are implementation-specific and often redundant, it has been preferred to leave them out of the class definition. If a record were to be read with additional fields, it could be argued that it does not strictly adhere to the OCSF schema that defined it.
- Profiles do not directly allow for any functionality that is not currently present in classes or categorization, particularly when the ability to “extend” a class is utilized.
- Profiles would allow for a diversification of schema to be referenced using the same class_id. That may allow for a broader appeal when queries are being filtered via class_id, but if the fields identified via the profile are relied upon in the queries, unexpected errors may result.
- It is unclear in the proposal whether or not the additional fields may be able to be included as “required” or if they would be included de facto as “optional”.