Open Cybersecurity Schema Framework

Author: Paul Agbabian
Date: June 2022
Status: RFC
Version: 1.4

Introduction to the Framework and Schema

This document describes the Open Cybersecurity Schema Framework (OCSF) and its taxonomy, including the cybersecurity schema. A taxonomy is a classification mechanism with conformation rules. The OCSF is a framework for creating schemas and it also delivers a security event schema built with the framework.

The framework is made up of a set of data types, a dictionary, a set of objects, and the taxonomy. Individual properties, or fields each have a specific validatable data type. More generally, fields and objects are referred to as attributes. The dictionary comprises all of the fields.

The scalar data types are defined on top of primitive data types such as strings, integers, floating point numbers and booleans. Object data type is a collection of attributes of any data type including other objects. Arrays support any of the data types. Examples of scalar data types are Timestamp, IP Address, MAC Address, Pathname, and User Name. Examples of Object data types are Process, Device, User, Certificate and File.

Some data types have constraints on their valid values or ranges, for example Enum integer types are constrained to a specific set of integer values. Enum integer typed attributes are an important part of the framework constructs and used in place of strings where possible to ensure consistency.

The dictionary of all available fields and their types and constraints and a set of standard objects are the building blocks of the framework. Appendix A and B describe the OCSF Guidelines and data types respectively.

Building a schema from the framework building blocks requires a taxonomy. The framework is not restricted to cybersecurity nor to events, however the initial focus of the framework has been a core schema for cybersecurity events.

OCSF is agnostic to storage format, data collection and ETL processes. However the schemas developed may have affinities to different implementations. The core schema for cybersecurity events is intended to be agnostic to implementations.

Personas

There are four personas that are users of the framework and/or the schema built with the framework.

The *author* persona is who creates or extends the schema. The *producer* persona is who generates events natively into the schema. The *mapper* persona is who translates or creates events from another source to the schema. The *analyst* persona is the end user who searches the data, writes rules or analytics against the schema, or creates reports from the schema.

For example, a vendor may write a translation from a native source format into the schema but also extend the schema to accommodate vendor specific attributes or operations. The vendor is operating as both the mapper and author personas. A SOC analyst that collects the data in a SIEM writes rules against the events and searches events during investigation. The SOC analyst is operating as the analyst persona. Finally, a vendor that emits events natively in OCSF form is a data producer.

Taxonomy Constructs

A *field* is a unique identifier name for a piece of data contained in OCSF. Each field also designates a corresponding validatable data type. Field names follow naming conventions based on their data types. An *object* is a collection of contextually related fields, possibly including other objects. It is also a data type in OCSF. An *attribute* is the more generic name for both fields and objects in OCSF. A field is a scalar attribute while an object is a complex attribute.

Attribute names are the unique identifier for each attribute. Attributes that are of an Enum type end with `_id`. Enum constant identifiers are integers from a set where each has a friendly name in a companion attribute of the same name, minus the suffix. Both must be populated, for validation, computational efficiency, and intuitive searching for the analyst persona.

There are four fundamental constructs of the OCSF taxonomy:

1. Event Class
2. Category
3. Profile
4. Extension

Each is defined with a unique identifier using a standard field attribute from the dictionary.

An event class has a unique `class_id` Enum attribute value which is the event class identifier. Event class Enum constant friendly names conform to conventions around their semantics, for example File Detection Event, Process Activity Event, User Session Activity Event.

Every event class has an activity, disposition or outcome, via the `activity_id` and `disposition_id` Enum attributes, constrained to the values appropriate for each event class.

Each event class is grouped by category and has a unique `category_id` Enum attribute value which is the category identifier. Event classes are grouped into categories for a few purposes: documentation convenience and search, reporting or access control. Category enums also have friendly names, as with all enums, such as System Activity, Network Activity, Findings, etc.

Profiles overlay additional related attributes into event classes and objects allowing for cross-category event class augmentation and filtering. Event classes register for profiles which can be optionally applied, or mixed into event classes and objects. For example, System Activity event classes may also emit malware detection or vulnerability information when an endpoint security product is the data source. Network Activity event classes from a host computer may carry the device, process and user associated with the activity.

Finally, the schema can be extended using the framework without modification of the core schema. New attributes, objects, event classes, categories and profiles are all available to extensions. Existing profiles can be applied to extensions, and new profiles can be applied to core event classes and objects as well as to other extensions.

Comparison with MITRE ATT&CK[1] Framework

The MITRE ATT&CK Framework is widely used in the cybersecurity domain. While the purpose and content type of the two frameworks are different, there are some similarities with OCSF's taxonomy that may be instructive to those with familiarity with ATT&CK.

Categories are similar to Tactics, which have unique IDs. Event Classes are similar to Techniques, which have unique IDs. Profiles are similar to Matrices[2], which have unique names. Type IDs are similar to Procedures which have unique IDs. Profiles can filter the Event Classes and Categories similar to how Matrices filter Techniques and Tactics.

Differences from MITRE ATT&CK are that in OCSF, Event Classes are in only one Category, while MITRE ATT&CK Techniques can be part of multiple Tactics. Similarly MITRE ATT&CK Procedures can be used in multiple Techniques. MITRE ATT&CK has Sub-techniques while OCSF does not have Sub-Event Classes.[3]

OCSF is open and extensible by vendors, and end customers while the content within MITRE ATT&CK is released by MITRE.

---

[1] MITRE ATT&CK: https://attack.mitre.org/

[2] MITRE ATT&CK Matrix: https://attack.mitre.org/matrices/enterprise/

[3] The internal source definition of an OCSF schema can be hierarchical but the resulting compiled schema does not expose sub classes.

Event Class

Events are represented by instances of Event Classes, which are a particular set of attributes and objects representing a log line or telemetry submission at a point in time. Event classes have semantics that describe what happened, either a particular activity, disposition or both.

Each event class has a unique `class_uid` attribute value which is the event class identifier. Event class friendly names populate the `class_name` attribute and are descriptive of some type of activity, such as File Access Activity or Process Activity.

The semantics of the class are defined by the specific activity, via the `activity_id` attribute, such as File Opened or Process Started. Other attributes of the class indicate the details such as the file name, or the process name. The unique combination of a `class_uid` and `activity_id` or `disposition_id` is represented by the `_type_uid` derived attribute.

A snippet of a File Activity event example is shown below.


```
{
   "category_uid": 1,
   "class_uid": 1004,
   "activity_id": 2,
   "event_name": "Endpoint File Activity:Read",
   "event_time": "1970-01-20T02:09:34.676997Z",
   "event_uid": 100400,
   "message": "File foobar.json opened",
   "severity_id": 1,
   "time": 1649374676992
}
```


It is the intent of the schema to allow for the mapping of any raw event to a single event class. This is achieved by careful design using composition rather than a multiple inheritance approach. In order to completely capture the information in a rich data source, many attributes may be required.

Unfortunately, aside from inconsistent naming and typing of extracted fields, driving the need for normalization, not every data source emits the same information for the same observed behavior. In the interest of consistency, accuracy and precision, the schema event classes specify which dictionary attributes are essential, or required, as not all are needed across different data sources. This is discussed further below in the Attribute Requirement Flags section.

Enum Attribute Convetions

By convention, every Enum type has two common values with integer value 0 for `Unknown` and -1 for `Other`.

If a source event has missing values that are required by the event class for that event, an `Unknown` value should be set for Enum types which is also the default.

If a mapped event field does not have a desired enumeration value corresponding to a value of the raw event, `Other` is used which indicates that a companion string field is populated with the field value.

By convention an Enum attribute name ends with `_id` for its integer value and its corresponding `Other` string field shares the name but without the suffix. For all other defined enumeration integer values, the label for the item also populates the string field. That is, both the integer value and the string field are always set. If the Enum attribute is required, then both the integer field and the string field are required.

Attribute Requirement Flags

Attributes in the context of an event class have a requirement flag, depending on the semantics of the event class. Attributes themselves do not have a requirement flag, only within the context of event classes.

The requirement flags are:

- Reserved
- Required
- Recommended
- Optional

Event class validation as well as attribute data type validation is enforced via the required attributes, in particular the classification attributes, which by necessity need to be kept to a minimum.

Reserved attributes are populated by the system and when defined within an event class will always be populated. Their names are prefixed with an underscore by convention.

Event classes are designed so that the most essential attributes are required, to give enough meaning and context to the information reported by the data source. If required attributes are not present for a particular event class, a default value is defined by the event class, usually `Unknown`.[4]

Recommended attributes should be populated but cannot be in all cases. They do not have default values. Optional attributes may be populated to add context and when data sources emit richer information.

[4] Required attributes that cannot be populated due to information missing from a data source must be carried with the event as *unknown* values - asserting that the information was missing.

Base Event Class Attributes

By convention, all event classes extend the Base Event event class. Attributes that can be present in any event class are termed Base Attributes.

Each event class has primary and optional attributes. Primary attributes are typically required, or recommended per event class, based on their use in each class. Optional base event attributes may be included in any event class, along with per event class specific optional attributes.

Examples of required base attributes are `class_uid`, `category_uid`, `severity_id`.

Examples of optional base attributes are, `start_time`, `end_time`, `count`, `duration`, `_unmapped`.

Special Base Attributes

There are a few base attributes that are worth calling out specifically. These are the `_unmapped` attribute, the `_raw_data` attribute and the `_type_uid` reserved attribute.

While most if not all fields from a raw event can be parsed and tokenized, not all are mapped to the schema. The fields that are not mapped may be included with the event in the optional `_unmapped` attribute.

The `_raw_data` reserved attribute holds the event data as received from the source. It is unparsed and represented as a String type.

The `_type_uid` reserved attribute is constructed by the system and is the combination of the event class of the event (`class_uid`) and its activity or disposition. It is unique across the schema hence it is a `_uid` suffix event rather than a `_id` suffix event. The `_type_uid` friendly name, `_type_name`, is a way of identifying the event in a more readable and complete way. For example Process Activity: Launched, or File Detection: Blocked.

Category

A Category organizes event classes that represent a particular domain. For example, a category can include event classes for different kinds of events that may be found in an access log, or audit log, or network and system events. Each category has a unique category_uid attribute value which is the category identifier. Category IDs also have category_name friendly name attributes, such as System Activity, Network Activity, Audit, etc.

An example of categories with some of their event classes is shown in the below table. Note, these are not final.

System Activity	Network Activity	Audit Activity	Findings	Cloud Activity
File Activity	Network Activity	Account Change	Security Finding	Cloud API
Folder Activity	HTTP Activity	Authentication		
Kernel Activity	DNS Activity	Authorization		
Memory Activity	DHCP Activity	Entity Change		
Module Activity	SSH Activity			
Peripheral Activity	RDP Activity			
Process Activity				
Scheduled Job Activity				
Registry Key Activity				
Registry Value Activity				
Resource Activity				

Finding the right granularity of categories is an important modeling topic. Categorization is weakly structural while event classification is strongly structural (i.e. it defines the particular attributes, their requirements, and specific Enum values for the event class).

Many events produced in a cloud platform can be classified as network activity. Similarly, many host system events include network activity. The key question to ask is, do the logs from these services and hosts provide the same context or information? Would there be a family of event classes that make sense in a single category? For example, does the NLB Access log provide context/info similar to a Flow log? Does network traffic from a host provide similar information to a firewall or router? Are they structured in the same fashion? Do they share attributes? Would we obscure the meaning of these logs if we normalize them under the same category? Would the resultant category make sense on its own or will it lose its contextual meaning all together?

Using profiles, some of these overlapping categorical scenarios can be handled without new partially redundant event classes.

Profile

Profiles are overlays on event classes, effectively a dynamic mix-in class of attributes and objects with their requirements and constraints.[5] While event classes specialize their category domain, a profile can augment existing event classes with a set of attributes independent of category. Multiple profiles can be added to an event class via an array of profile values in the `profiles` attribute. This mix-in 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 `profiles` attribute 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. A cloud provider profile could mix-in cloud platform specific information onto network activity events.

Proposals for three built-in profiles for `Malware`, `Host` and `User` are shown in the below table with their attributes.

Malware Profile	**Host Profile**	**User Profile**
`disposition_id / disposition`	`device`	`user`
`attacks`	`connection_info`	`is_user_present`
`cvssv2`	`actor_process`	`user_entities`
`malware / other_malware`		`accounts`
`quarantine_uid`		`user_result`

Other profiles could be product oriented, such as Firewall, IDS, VA, DLP 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 services log events with an ARN (AWS Resource Name) and an AWS IAM Account. An AWS specific profile can be added to any event class or category of classes that includes arn and IAM account attributes. Splunk Technical Add-ons would define a profile that

[5] Refer to Proposal 3: Profiles in the document OCSF Schema Collaboration: Initial Decisions

would be added to all events with Splunk's standard source, sourcetype, host, attributes.Profile Application Examples

Using example categories and event classes from a preceding section, examples of how profiles might be applied to event classes are shown below.

System Activity

The following **would** all include the Host profile and **may** include the Malware profile:
File Activity
Folder Activity
Kernel Activity
Memory Activity
Module Activity
Peripheral Activity
Process Activity
Resource Activity
Scheduled Job Activity

Windows Activity

The following **would** include the Host profile and **may** include the Malware profile:
Registry Key Activity
Registry Value Activity

The following **would** include the Host profile:

Windows Event

Network Activity

The following **may** include the Host profile and **may** include the Malware profile:
DNS Activity
HTTP Activity
Network Activity

Audit Activity

The following **would** include the User profile, **may** include the Host profile and **would not** include the Malware profile:
Account Change
Authentication

Authorization
Entity Activity

Personas and Profiles

The personas called out in an earlier section, producer, author, mapper, analyst, all can consider the profile from a different perspective.

Producers, who can also be authors, can add profiles to their events when the events will include the additional information the profile adds. For example a vendor may have certain system attributes that are added via an extension profile. A network vendor that can detect malware would apply the Malware profile to their events. An endpoint security vendor can apply the Host, User and Malware profile to network events.

Authors define profiles, and the profiles are applicable to specific classes, objects or categories.

Mappers can add the profile ID and associated attributes to specific events mapped to logs in much the same way producers would apply profiles.

Analysts, e.g. end users, can use the browser to select applicable profiles at the class level. They can use the profile identifier in queries for hunting, and can use the profile identifiers for analytics and reporting. For example, show all malware alerts across any category and class.

Extensions

Extensions are additional categories, event classes, attributes, objects or profiles. The Open Cybersecurity Schema Framework can be extended by adding new attributes, objects, categories and event classes. A schema is the aggregation of core schema entities and extensions.

Extensions allow a particular vendor or customer to create a new schema or augment an existing schema. Extensions can also be used to factor out non-essential schema domains keeping a schema small. Extensions use the framework in the same way as a new schema, optionally creating categories, profiles or event classes from the dictionary. Extensions can add new attributes to the dictionary, including new objects. As with categories, event classes and profiles, extensions have unique IDs within the framework as well as versioning.

Examples of new experimental categories, new event classes that contain some new attributes and objects are shown in the table below with a `Dev` extension superscript convention. In the example, extension classes were added to the core Findings category, and three extension categories were added, Policy, Remediation and Diagnostic, with extension classes.

Findings	**Policy**[Dev]	**Remediation**[Dev]	**Diagnostic**[Dev]
Incident Creation[Dev]	Clipboard Content Protection[Dev]	File Remediation[Dev]	CPU Usage[Dev]
Incident Associate[Dev]	Compliance[Dev]	Folder Remediation[Dev]	Memory Usage[Dev]
Incident Closure[Dev]	Compliance Scan[Dev]	Unsuccessful Remediation[Dev]	Status[Dev]
Incident Update[Dev]	Content Protection[Dev]	Startup Application Remediation[Dev]	Throughput[Dev]
Email Delivery Finding[Dev]	Information Protection[Dev]	User Session Remediation[Dev]	

To extend the schema create a new directory in the `schema/extensions` directory. The directory structure is the same as the top level schema directory and it may contain the following files and subdirectories:

categories.json	Create it to define a new event category to reserve a range of class IDs.
dictionary.json	Create it to define new attributes.
events/	Create it to define new event classes.
objects/	Create it to define new objects.

Appendix A - Guidelines and Conventions

The Open Cybersecurity Schema Framework (OCSF) guidelines and conventions.

Attribute Levels

The event schema defines *Core*, *Optional*, and *Reserved* attributes.

Core Attributes

Attributes that are most common across all use cases are defined as core attributes. The core attributes are marked as Required or Recommended.

Optional Attributes

Optional attributes may apply to more narrow use cases, or may be more open to interpretation depending on the use case. The optional attributes are marked as Optional.

Reserved Attributes

Reserved attributes are set by the logging system and must not be used in the event data. The reserved attributes are marked as Reserved.

Guidelines for attribute names

- Attribute names must be a valid UTF-8 sequence.
- Attribute names must be all lower case.
- Combine words using underscore.
- No special characters except underscore.
- Use present tense unless the attribute describes historical information.
- Use singular and plural names properly to reflect the field content.
  For example, use `events_per_sec` rather than `event_per_sec`.
- When an attribute represents multiple entities, the attribute name should be pluralized and the value type should be an array.
  Example: `process.loaded_modules` includes multiple values -- a loaded module names list.
- Avoid repetition of words.
  Example: `host.host_ip` should be `host.ip`.
- Avoid abbreviations when possible.
  Some exceptions can be made for well-accepted abbreviations. Example: `ip`, or names such as `os`, `geo`.
- For vendor extensions to the dictionary, prefix attribute names with a 3-letter moniker in order to avoid name collisions. Example: `aws_finding, spk_context_ids.`

Appendix B - Data Types

The predefined data types. The data type of a value specifies what kind of data that value can have. Note type^{O} denotes an observable type. _t attributes in parentheses denote internal JSON schema type notation.

Attribute	Base Type	Constraints	Description
Boolean (boolean_t)		false, true	Boolean data type.
Email Address° (email_t)	String	^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$	Email address type. For example: john_doe@example.com.
File Hash° (hash_t)	String	Max length: 64	File Hash value. A unique value that corresponds to the content of the (le.
File Name° (file_name_t)	String	^[a-zA-Z0-9._ -]+$	File name. For example: /text-file.txt.
Float (float_t)			Real Floating-point data type.
Hostname° (hostname_t)	String	^(([a-zA-Z0-9]|[a-zA-Z0-9][a-zA-Z0-9\-]*[a-zA-Z0-9])\.)*([A-Za-z0-9]|[A-Za-z0-9][A-Za-z0-9\-]*[A-Za-z0-9])$	A unique name assigned to a device that is connected to a speci(c computer network. A domain name in general is an Internet address that can be resolved through the Domain Name System (DNS). For example: r2- d2.example.com.

Attribute	Base Type	Constraints	Description
IP Address[o] (ip_t)	String	Max length: 40 /^(?>(?>([a-f0-9]{1,4})(?>:(?1)){7}|(?!(?:.*[a-f0-9](?>:|$)){8,})((?1)(?>:(?1)){0,6})?::(?2)?)\|(?>(?>(?1)(?>:(?1)){5}:\|(?!(?:.*[a-f0-9]:){6,})(?3)?::(?>((?1)(?>:(?1)){0,4}):)?(25[0-5]\|2[0-4][0-9]\|1[0-9]{2}\|[1-9]?[0-9])(?>\.(?4)){3}))$/iD	Internet Protocol address (IP address), in either IPv4 or IPv6 format.
IP Port (port_t)	Integer	0-65,535	IP TCP/UDP port number. For example: 80 or 22.
Integer (integer_t)	Integer		Basic signed integer data type.
JSON (json_t)	String		Embedded JSON value. A value can be a string, or a number, or true or false or null, or an object or an array. These structures can be nested. See www.json.org.
Long long_t	Long		8-byte long, signed integer data type.
MAC Address[o] (mac_t)	String	Max length: 32 ^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$	Media Access Control (MAC) address. For example: 18:36:F3:98:4F:9A.
Object object_t			Object is an unordered set of name/value pairs. For