



Absolute Beginners Guide to OCSF

Contributed by: Aurora Starita



Table of Contents

What is OCSF? 1

OCSF Explained (Simply) 2

Why Use OCSF? 3

Your Formal Invitation to OCSF Events 4

A View From the Top: The Grand Schema of Cybersecurity Things 6

Extensibly Yours 10

Ingredients for a Great Event 11

Where To Go From Here 17



What is OCSF?

This comprehensive open cybersecurity schema framework introduction is written to be extremely beginner-friendly.

It's the first OCSF introduction to exist, to my knowledge. My goal for this absolute beginner's guide is to give anyone at any skill level enough of an overview of the framework that they will be able to mostly grok the OCSF schema browser as well as be able to follow along with our slightly more advanced blogs and [OCSF tutorials](#).

We all start our journey into anything cybersecurity from different places, already knowing different things.

To cover as many types of beginners as possible, I will assume you know a decent amount about cybersecurity events, which I won't explain in any detail, but very little about data organization or programming concepts, which I may explain in insulting levels of detail.

If you do have some experience in those areas, you're still invited to follow along for a calm sail through this OCSF primer, but you may prefer the slightly rougher (and faster) seas of learning OCSF while also learning [the basics of OCSF mapping](#).

By the end of this guide you should understand what OCSF means, what OCSF does, why organizations choose OCSF, and even a little bit "how to do OCSF".

Alright. **So what is OCSF?**



OCSF Explained (Simply)

The **Open Cybersecurity Schema Framework (OCSF)** is a standard for describing events—things that happen within an organization's IT system and are then logged somewhere—that might be of interest to cybersecurity professionals.

These events might be things that are signs of the system being compromised somewhere or they might be everyday activity. **OCSF just helps you make a blueprint (AKA a schema)** for organizing all the data to be logged or mapped in order to be used and it's up to people and tools to come up with a higher level meaning of any particular event.

The O in OCSF is for “open”, meaning this framework is freely available to everyone and not designed with any one tool or company in mind—in fact, it's designed with the knowledge that organizations have logs from many sources and so provides a way to **unify diverse data**.

Without something like OCSF, the data from an organization's many cybersecurity tools (estimates vary but it's about 70 tools on average for an enterprise company) basically all speak different languages, which is a hassle to normalize and standardize. **OCSF creates a common language for everyone.**

OCSF is more than just an arbitrary standard; care was taken in its design to make retrieving and searching records of events faster and easier on both the people requesting the information and the computers doing the tasks to fulfill the request.

OCSF is defined in JSON, a programming language-independent data format that is widely used and that many programming languages have built-in (or very easy to get) tools for dealing with.

And there isn't one The Schema™. Rather, OCSF is, as the name suggests, a framework for creating schemas (or “schemata” if that's what you're into—no judgment). OCSF is designed to be extended to apply to use cases the project contributors haven't even thought of yet.

We can all have cybersecurity schemas that work for our data!

In a nutshell, OCSF recognizes:

- There's a lot of security-relevant data out there.
- Normalizing the data coming from different tools without a standard plan takes up a lot of time and that kind of sucks for everyone that needs to use that data.
- Security data can be organized in a way that makes connecting the dots between two or more separate events faster and easier.

Why Use OCSF?

How do you use OCSF? Depends who you are. Different roles are going to care about and interact with OCSF in different ways. In [Understanding the Open Cybersecurity Schema Framework](#)—a seminal text on the subject by the distinguished godfather of OCSF himself, Paul Agbabian—four personas are described:

Authors

Authors better know OCSF inside and out, because they are the ones who actually create or extend schemas.

Producers

Producers are the people in charge of generating events into OCSF natively (or as a translation) so that the end user of their products can benefit. Producers are typically on the side of companies that make cybersecurity tools that produce data to be consumed by cybersecurity customers. Producers need a good technical grasp of OCSF details.

Mappers

Mappers take the data from sources that aren't providing natively OCSF-formatted logs and translate it. Mappers are typically on the cybersecurity tool user side (with the analysts) but have roughly the same work cut out for them as producers do. Query ❤️'s Mappers, but in the glorious future as OCSF adoption grows, there will be way more producers than mappers.

Analysts

Analysts are the end users of the schema and OCSF-formatted data. Analysts benefit from OCSF when they search OCSF-formatted data, write rules against an OCSF-derived schema, or create reports. Making life easier for analysts by normalizing and standardizing data is the reason that authors, producers, and mappers are even bothering to adopt OCSF in the first place. So analysts don't necessarily need to know anything about OCSF at all.

Full disclosure: Like OCSF itself, Query is here for the analysts. We've mapped a number of tools' outputs into OCSF so that we can provide awesome and powerful federated searches of security data, and we will continue to do that with more and more tools.

But when these tools start producing OCSF-formatted outputs themselves, we will become even more awesome and powerful. So you could say we are very interested in wide OCSF adoption.

So if you're a future mapper, producer, or author—or an analyst who likes to pop open the hood and see what's going on—read on.

A Note On Conventions Here

I've capitalized some words of importance because they are important. Most texts about OCSF don't capitalize these words.

My first idea was to capitalize all OCSF-specific words, all the time, but this looked extremely cluttered, dense, overly-technical, and German, which was not my intention. I settled on capitalizing just the words that are most title-like.

Your Formal Invitation to OCSF Events

If there's a "basic unit" of OCSF it has to be the **event**, specifically the **Base Event**, which we'll cover in a second. An event is exactly what it sounds like: something that occurs that's cybersecurity related in some way.

Things like:

- Someone logs in
- Someone logs out
- Some file is read
- Some security tool finds something and issues an alert
- A drone is discovered in your airspace (coming soon)

Each of those events has some important information you'd like to know.

Such as:

- Who logged in?
- What was their IP address?
- When did this happen?

The answers to these kinds of questions show up under an OCSF event as **attributes**, which we'll get more in-depth about in the next white paper in this series.

There are a lot of different kinds of events in cybersecurity, but each kind of event will usually have roughly the same kinds of things (**attributes**) that are logged and that you could possibly know about them.

Luckily for us all, every common kind of cybersecurity event and the attributes that describe it are already blueprinted up for you by OCSF. Each of these forever-reusable event blueprints is called an **event class**.

If you're not familiar with classes in programming, just think of them as templates for what properties a bunch of things you're going to make in the future should have.



Predefined Event Classes in OCSF

The latest major version as of this writing, OCSF v1.3, describes 64 specific cybersecurity-relevant event classes.

Here are a few examples:

Name	Description
File System Activity	File System Activity events report when a process performs an action on a file or folder.
Vulnerability Activity	The Vulnerability Finding event is a notification about weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.
Email Activity	Email events report activities of emails.
Authentication	Authentication events report authentication session activities such as user attempts a logon or logoff, successfully or otherwise.

The Base Event Class

OCSF provides a generic Base Event class that can be used to describe funky random types of events that don't relate well to the event classes predefined by OCSF—but that's possibly selling the [Base Event](#) class a little short there.

The Base Event class is extremely foundational to OCSF. It is the most generic event class there is and all of the other event classes derive from it, which is to say that every single event class has the same attributes as the Base Event plus additional attributes that are specific to the event class.

Now that you know what an event class is, **you should know the true definition of "event"**. An event is an instance (singular occurrence) of an event class, where we actually know the specific when, where, how, and at what time one specific something happened. If the event class is a heart-shaped cookie cutter then an event is the real, edible heart-shaped cookie.

Take note: People will sometimes (often) say **"the event"** when they mean **"the event class"** when discussing OCSF, particularly when the discussion is about OCSF itself, which is by its very nature a very abstract kind of discussion.

One very common example is when people are talking about **"the Base Event"**, like I did just a few paragraphs ago and probably will again, they're almost always talking about the **"Base Event class"**.

We're going to get into a lot more detail about event classes and the stuff inside them a little later. Before we touch that, we should zoom out and get a greater sense of how the framework itself is built and organized.

A View From the Top: The Grand Schema of Cybersecurity Things

This is a screenshot of the OCSF schema browser when you first land on it. It is an interactive visual representation of the **core schema**, which is all the hard work OCSF contributors have already done for everyone that can be generic enough to be applied across many cybersecurity situations and tools.

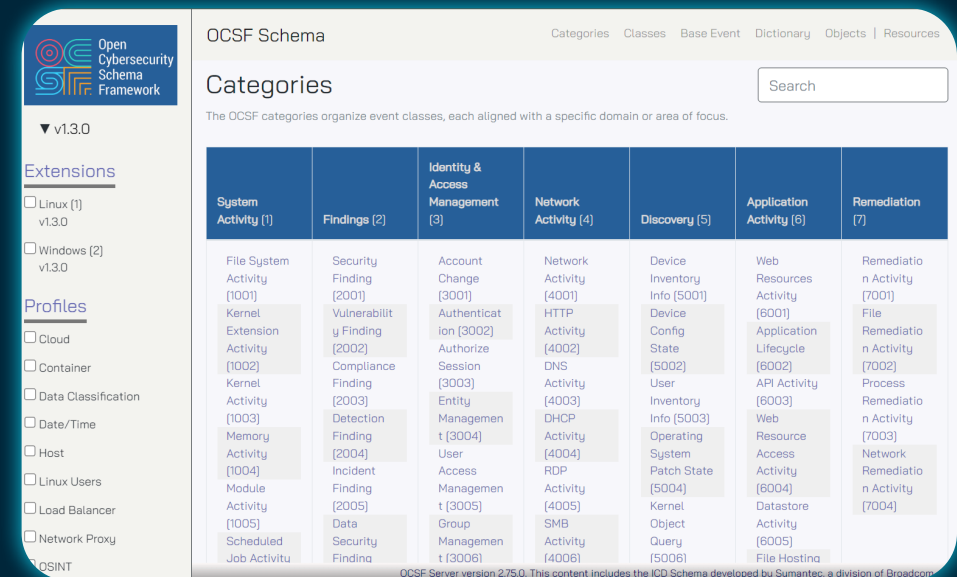
So, a little recap: In OCSF the **framework** is a blueprint for making **schemas** which are themselves a blueprint for organizing events which are each made from an **event class** blueprint.

So sorry to do this, but there's also a **metaschema**. The metaschema is what's laid out when you click through different event classes on the [OCSF schema browser](#) and includes all of the little details about what data type a particular attribute has to be and whether or not that attribute is required or optional.

Ultimately, it's just the documentation to follow so you make an OCSF schema and not something else. It makes for excellent guidance as you work with OCSF for your own data.

You'll learn in our [Definitive Guide to Open Cybersecurity Schema Framework \(OCSF\) Mapping](#) blog that the metaschema can be more of an ideal than a hard law, depending on what you're trying to accomplish.

Alright, for now we're just going to stay on that top level view of that OCSF schema browser link. **What are we looking at here?**



The screenshot shows the OCSF Schema browser interface. On the left, there are navigation menus for 'Extensions' (Linux, Windows, Cloud, Container, Data Classification, Date/Time, Host, Linux Users, Load Balancer, Network Proxy, OSINT) and 'Profiles'. The main content area is titled 'OCSF Schema' and 'Categories'. It includes a search bar and a table of categories.

OCSF Schema

Categories | Classes | Base Event | Dictionary | Objects | Resources

Categories

The OCSF categories organize event classes, each aligned with a specific domain or area of focus.

System Activity (1)	Findings (2)	Identity & Access Management (3)	Network Activity (4)	Discovery (5)	Application Activity (6)	Remediation (7)
File System Activity [1001]	Security Finding [2001]	Account Change [3001]	Network Activity [4001]	Device Inventory Info [5001]	Web Resources Activity [6001]	Remediation Activity [7001]
Kernel Extension Activity [1002]	Vulnerability Finding [2002]	Authentication [3002]	HTTP Activity [4002]	Device Config State [5002]	Application Lifecycle [6002]	Remediation Activity [7002]
Kernel Activity [1003]	Compliance Finding [2003]	Session [3003]	DNS Activity [4003]	User Inventory Info [5003]	API Activity [6003]	Process Remediation Activity [7003]
Memory Activity [1004]	Detection Finding [2004]	Entity Management [3004]	DHCP Activity [4004]	Operating System Patch State [5004]	Web Resource Access Activity [6004]	Network Remediation Activity [7004]
Module Activity [1005]	Incident Finding [2005]	User Access Management [3005]	RDP Activity [4005]	Kernel Object Query [5005]	Datastore Activity [6005]	File Hosting Activity [7005]
Scheduled Job Activity [1006]	Data Security Finding [2006]	Group Management [3006]	SMB Activity [4006]			

OCSF Server version 2.75.0. This content includes the ICD Schema developed by Symantec, a division of Broadcom.

Categories

With the exception of the generic Base Event, which can be under any category or none at all, all event classes fall under one of 7 (soon to be 8) top-level [categories](#).

Why? A few reasons:

- Easier for you to find what you're looking for on the schema browser.
- Specific definitions of things can be extended across all event classes in a category.
- Order must prevail over chaos

Note: Links to specific categories and event classes in the schema browser are going to be for the **latest version of OCSF as of this writing (v1.3.0)**.

If you're reading this in the distant future, **there is a version picker in the top left-hand corner under the OCSF logo** that you can use to change the version to whatever the current version is.

	Category Name	Example Category event classes
1	System Activity	File system activity [1001], Memory activity [1004]
2	Findings	Security finding [2001], Compliance finding [2003]
3	Identity & Access Management	Authentication [3002], User Access Management [3005]
4	Network Activity	HTTP Activity [4002], Email Activity [4009]
5	Discovery	File Query [5007], Software Inventory Info [5020]
6	Application Activity	Web Resources Activity [6001], API Activity [6003]
7	Remediation	File Remediation Activity [7002], Network Remediation Activity [7004]


Categories *(cont.)*

You may also notice some numbers next to those event class names. Wherever possible, **OCSF makes use of integers** (you know, “whole numbers”) for the benefit of computers.

English names of things are used **only secondarily** to help you as a human. We’ll talk more about data types later.

Oh, hey, remember when I said that each event class has the **same attributes** as the Base Event **plus additional attributes** that are specific to the event class?

Well, I sort of lied. **There’s another step...**



Open
Cybersecurity
Schema
Framework

▼ v1.3.0

v1.0.0

v1.0.0-rc.2

v1.0.0-rc.3

v1.1.0

v1.2.0

v1.3.0

v1.4.0-dev

OCSF Schema

Categories

The OCSF categories organize ev

System Activity [1]	Finding
File System Activity [1001]	Secu
Kernel Extension Activity [1002]	Findi
Kernel Activity [1003]	Vulne
	Findi
	Comp
	Findi
	Det

OCSF Secret Menu: Category Events

It is true that any given event class is just some more specific attributes added to the Base Event class, but the event class doesn't usually gain the Base Event attributes directly. All of the event classes under any given category are very likely to share some attributes.

So what (usually) happens is first, something called a **Category Event class** extends the Base Event and then the event class extends the Category Event class.

Put another way: the Category Event class is the Base Event class + some extra Category-specific attributes. The final event class is the Category Event class + some extra event class-specific attributes.

The Category Event class is very real but you can't see it in the schema browser, only in the actual JSON files.

So, again, it goes: the event class inherits the attributes of the Category Event class which includes all of the attributes the Category Event inherited from the Base Event class.

Is this going to be super important for anything you're likely to get up to in OCSF?

Maybe so, maybe not.

I'm only giving you these specifics now so that if you decide to browse through the JSON files in the OCSF schema repository, you know what's going on.

When you're rifling around and you look at, for instance, `network_activity.json` and it seems to be missing attributes you'd expect, so you go and look at `base_event.json` and they're not there either, that's because those attributes are probably described in `network.json`, which describes that middleman, the Network Category Event class.

Now that I've told the honest truth about Category Events and how OCSF is extensible within the core schema by allowing schema parts to reference other schema parts, let's talk about how **OCSF is designed to extend out from the core schema** so it can cover everyone's particular needs.



Extensibly Yours

The core schema can be extended for making very specific-to-your-particular-data schemas in a standard way that will play nice with others, which is why we see **extensions and profiles in the left column of the schema browser**.

Some extensions and profiles are already provided within the core schema, which are great in their own right, but are **also helpful studies in how to make your own**. That said, extending OCSF is a bit of an advanced topic so we're going to keep it short for now.

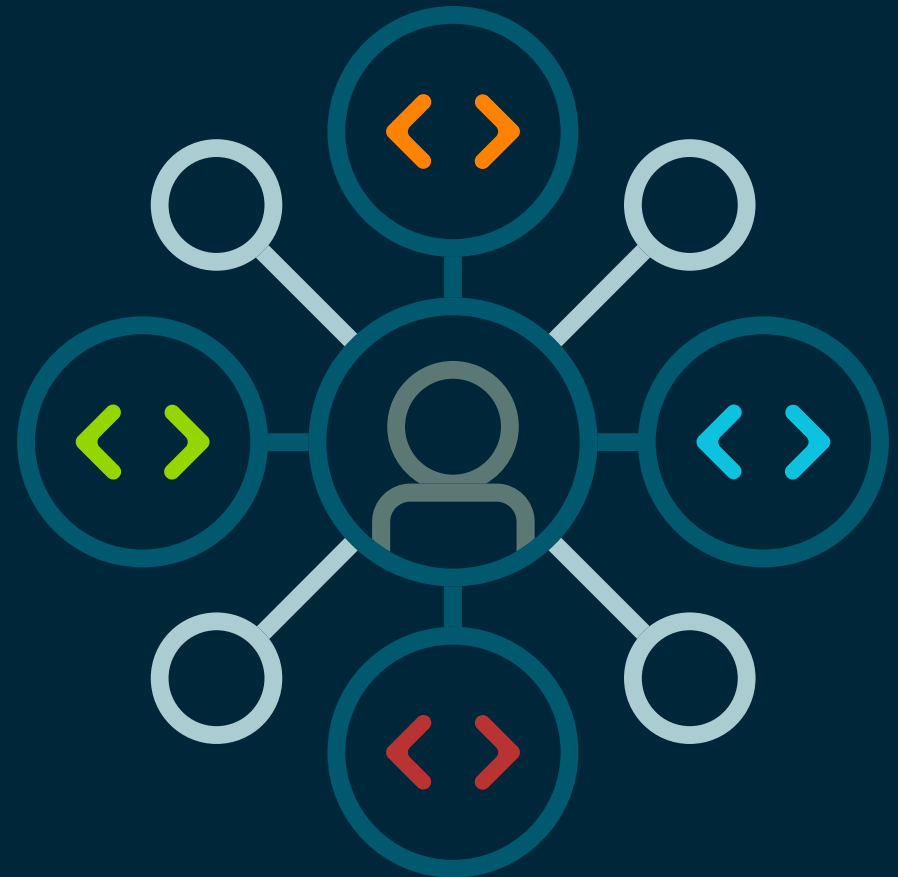
Extensions

Extensions let you **add your own event classes**. There are 64 event classes already defined but maybe you've got your own thing going on. That's cool and OCSF is here for it.

Profiles

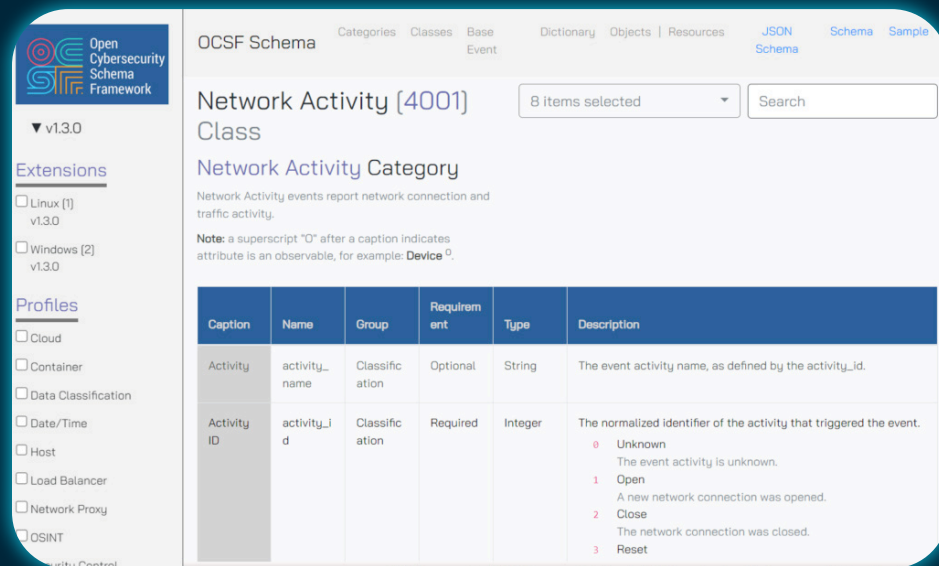
The purpose of profiles is to be able to **optionally add certain extra attributes to certain event classes**. You can use profiles when you want to extend an event class in the core schema instead of using an extension to create a whole new one.

Short and sweet. Now that we know what we're seeing on the schema browser landing page, let's go see what's underneath event classes.



Ingredients for a Great Event

So we're off the front page of the OCSF schema browser now and we're checking out some event classes in detail. We'll choose the Network Activity [4001] event class.



In the top right corner, **the three blue links** let us see some things that are all helpful for mapping but we won't worry about too much right now.

- **JSON Schema** shows us the raw JSON for this event class.
- **Schema** shows us a sample of the schema for this event class.
- **Sample** shows us what an actual event (not class!) with actual data would look like.

Below that we see a big table with **many rows of attributes** and six columns that each tell us something about each attribute.

- **Caption** is for us humans. It's a very short, very readable description of the attribute.
- **Name** is the attribute's official, machine-friendly name in the schema. It's used as a key (something to look up) that points to a value (the specific information for that attribute).
- **Group** tells us what this attribute is broadly about.
- **Classification** is for attributes related to the framework itself, rather than specific event details.
- **Occurrence** means this attribute has something to do with time.
- **Primary** means to expect this attribute to exist in all use cases of this event class.
- **Context** means this attribute is used to add enrichment information that is not typically vitally important to understanding the event.
- **Requirement** tells us how necessary providing data for this attribute is (you'll learn more about that in our blogs about mapping).
- **Type** tells us the kind of data we're dealing with—more on this shortly.
- **Description** provides extra details, like what specific values for the attribute mean, which we'll get into when we discuss enumerations.

Let's talk about type so we can discuss that second description in the screenshot a little better.

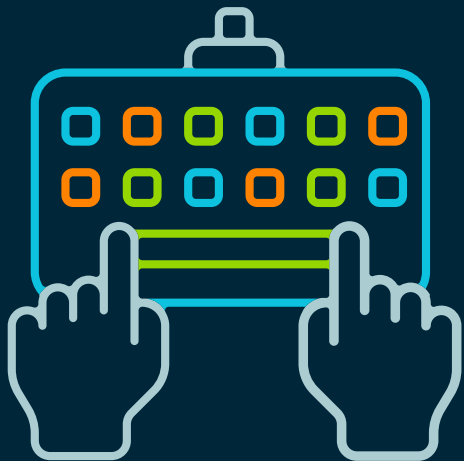
What's Your Type?

It all becomes 1s and 0s eventually, but functioning programs rely very heavily on the computer knowing **what kind of information it's dealing with** at all times.

When we write programs, we must (usually very explicitly) tell the computer what type of data has been or will be given to it, which in turn tells it what kind of things can and can't be done with it and how much space in memory it needs to reserve for it.

As OCSF producers and mappers, **we need to know what kind of information is expected by the schema for each attribute** so we can provide it in the right format and maintain standardization and normalization because that's the whole freaking point of OCSF.

So let's quickly go over the types you're likely to encounter as you look through event classes on the OCSF schema browser.



Scalar Types

For our purposes, scalar means **"not an object"** but we haven't gone over objects yet so just pocket that for now. Here are some examples of scalar types:

String is just a sequence of characters. The string data type is used to store text information and in most programming languages you put quotation marks around strings so it's clear what's part of a string and what's actual program instructions.

Examples:

```
"Cheeseburger"
```

```
"12342375423918" //(Since it's in quotes, the  
computer treats it as text, not a numerical value.)
```

Boolean is for true/false, yes/no kind of information. Boolean is often used for decision-making.

Float (sometimes called "double" elsewhere) is a number with a decimal point and numbers after it.

Examples:

```
1234.567
```

```
20.0
```

Scalar Types *(cont.)*

Integer just means whole numbers, as in numbers without a decimal point or anything after it. OK, technically, whole numbers are only positive numbers (and zero) and integers can also be negative numbers. Long is for really large integers and it's a separate type because there are limits within programming languages to the longest value a regular integer can hold.

Enum or **enumeration** is a special kind of integer, where each integer represents an option, rather than a quantity, and those options are defined by the user rather than the programming language. As programmers and mappers, we're concerned with the enumerations already defined by OCSF so we're keeping things standard. As authors we might get to invent our own when we create extension event classes.

If you've ever called a big company and dealt with their menu options—press 1 for billing, press 2 for sales, press 3 to cycle through an endless loop of automated messages without ever speaking to a human—you are already intimately familiar with enumeration.

Timestamp is for data/time values. The standard for an OCSF timestamp is the Unix timestamp, which is the number of seconds since midnight on January 1st, 1970 (also known as the Unix epoch).

Activity ID	activity_id	Classification	Required	Integer	The normalized identifier of the activity that triggered the event.
					0 Unknown The event activity is unknown.
					1 Open A new network connection was opened.
					2 Close The network connection was closed.
					3 Reset The network connection was abnormally terminated or closed by a middle device like firewalls.
					4 Fail The network connection failed. For example a connection timeout or no route to host.
					5 Refuse The network connection was refused. For example an attempt to connect to a server port which is not open.
					6 Traffic Network traffic report.
					7 Listen A network endpoint began listening for new network connections.
					99 Other The event activity is not mapped. See the <i>activity_name</i> attribute, which contains a data source specific value.

Here's the full view of that **Activity ID attribute** for the **Network Activity class** and you can see it lists integer as the type but in the description, a predefined list of available integers and what they represent is given.

Complex Types (Objects)

Object is for something that represents a noun (person, place, thing, or idea) and includes lots of information about what that noun is made up of. Objects are user-defined like enumerations.

You can think of objects like this: A car is one thing (“a car”) but there are all kinds of ways we can describe what the car looks like, what it does, what it’s made from. So we might create an object `car` that has attributes like `tires`, `motor`, `paint_color`, `weight`, etc. Now you might be thinking “paint color and weight seem like they could be described with scalar types, but tires and motor seem like objects themselves”. Damn right.

Objects can refer to other objects—which can refer to other objects themselves—but eventually you’ll get down to an object that’s entirely made of scalar attributes.

Objects are very similar to events in that what I’m really talking about here is an object class. An object class is a model or blueprint (yup, another blueprint) that describes the kinds of properties a particular object can have.

“An object” is an instance (singular occurrence) of an object class. So `car` is an object class (with attributes like `paint_color`) and `my_honda_odyssey` is an object (where `paint_color` is “blue”). But will you frequently find people saying “object” when they mean “object class”? Yes. Sorry.

Just as many event classes are already defined within the OCSF core schema, so too are many objects, all of which you can see in the schema browser under “Objects”.



Other Types

Array is for grouping a bunch of things of the same type (like integer, string, object, etc) but with just one reference to the entire group of things. Think of it like instead of saying “I need bananas, milk, bread, carrots, cereal, and peanut butter”, you say “here’s a grocery list” and then your personal shopper can take the whole list from you and look at each item when they need to.

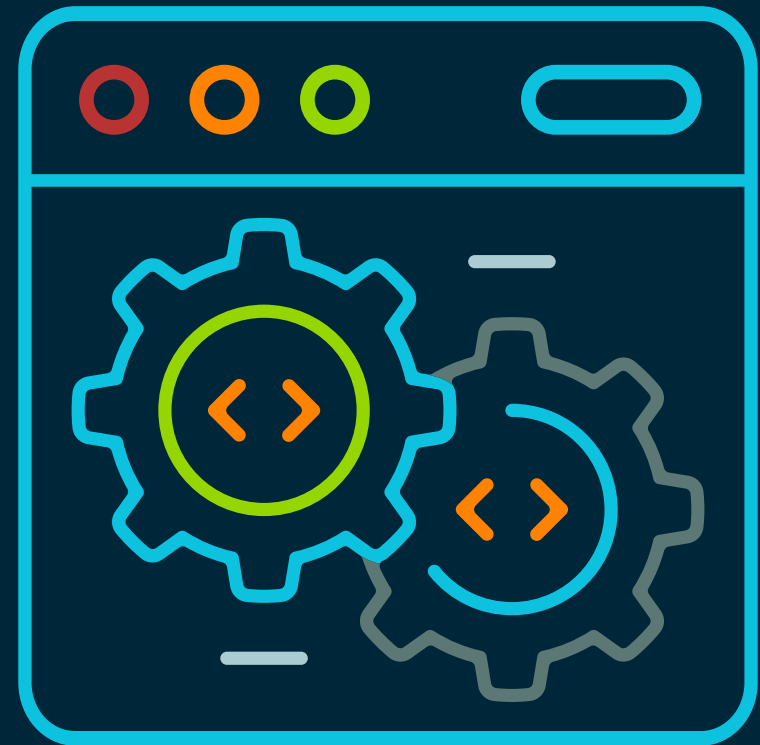
By the way, arrays are considered a ‘compound’ data type (not scalar), but in OCSF, ‘complex’ specifically refers to objects. In programming, arrays and lists aren’t exactly the same thing, but the difference doesn’t matter for now.

If it helps, you can think of an array as a list in the everyday sense, like with the grocery list example. You won’t be alone; you’ll often see array-type attributes described as a ‘list’ in their descriptions in the schema browser.

Map (or “dictionary” in some programming languages) is a way to store pairs of related information, known as key-value pairs. Think of it as a very simple database where each unique “key” has a specific “value” linked to it, and you can quickly look up values by their keys. It’s a lot like the contact list in your phone. You can look up a name (**key**) and get a number (**value**).

You won’t see map as a type for any event class attribute in the schema browser, but it’s still a fundamental concept that helps as you map data and navigate OCSF.

Null isn’t a data type itself but a concept representing nothing or the absence of a value. It’s useful for cases when information is intentionally left blank.



Observables

An observable is a “pivot element”, something that you might be interested in seeing no matter what kind of event it shows up in. **At Query, we call them “entities”.**

Imagine you’ve got information coming from a bunch of sources—all OCSF formatted, of course. Say one source triggers an alert about a suspicious email. Looking into the details of the alert, you see it is connected to a particular IP address. You would naturally want to know if that IP address shows up anywhere else in your other sources of information—like logs, alerts, or network activity data—to see if there’s a pattern or related threat.

If you dig through the event classes and object attributes, you’ll find that a number of different attributes are for IP addresses, because an IP address will have different significance or meaning in different kinds of events.

Instead of having to remember all of those different places where an IP address might be, observables let us say ahead of time that an IP address is an IP address, and we’re interested in being able to find any IP address no matter what type of event it’s a part of.

The Base Event includes the attribute observables which is an array of observable objects. Because it’s in the Base Event class, **it’s a part of every event class**. Observables are very cool and we’ll go into a lot more detail on them in a future blog.

Here are the things we can observe:

Value	The Observable Value Type Identifier	Value	The Observable Value Type Identifier
0	Unknown	16	HTTP User-Agent
1	Hostname	17	CWE Object: uid
2	IP Address	18	CVE Object: uid
3	MAC Address	19	User Credential ID
4	User Name	20	Endpoint
5	Email Address	21	User
6	URL String	22	Email
7	File Name	23	Uniform Resource Locator
8	Hash	24	File
9	Process Name	25	Process
10	Resource UID (unique identifier)	26	Geo Location
11	Port	27	Container
12	Subnet	28	Registry Key
13	Command Line	29	Registry Value
14	Country	30	Fingerprint
15	Process ID	99	Other

Where To Go From Here

Welp (Midwestern for “it’s time to get going”), that’s all for this OCSF explained. We’ve covered the Base Event class, event classes and their categories, data types including objects, and even a little bit about extensions and observables.

Now that you have a better grasp of some of the basic concepts underlying OCSF, the world of OCSF is yours to explore.

I recommend you:

- Explore the [OCSF schema browser](#).
- Look through the JSON files in the [OCSF/schema repository](#), especially those in the events and objects folders, as well as dictionary.JSON which defines all attributes.
- Read [Understanding the Open Cybersecurity Schema Framework](#) by Paul Agbabian.
- Watch Neal Bridges, Query CISO, and Jeremy Fisher, Query CTO, [discuss OCSF](#) (which includes a performance of Jeremy’s OCSF song).
- Check out our [quick OCSF walkthrough](#) of a real world example.
- Learn to map with our [OCSF mapping blog](#).



Query: Making Open Federated Search for Security a Reality

Query aims to deliver visibility into all relevant data for security teams. We provide a **federated search solution** that allows operators to **access data at the source** and in your data lakes, creating opportunities for more nimble and cost efficient data storage architectures.

Our customers are using Query to expand visibility for security investigations, threat hunting, and incident response. They are drastically **reducing the time and complexity** of repetitive search tasks and **improving outcomes for investigations**. Expose your security data with Query.

Ready to **expedite your security investigations** with open federated search for security?

For more information visit: www.query.ai