Media Over QUIC
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

S. Nandakumar
C. Jennings
Cisco Systems, Inc.

20 October 2025

Mapping Open Cybersecurity Schema Framework Events to Media over QUIC Transport
draft-nandakumar-moq-ocsf-mapping-01

Abstract

This document defines a mapping specification for representing Open Cybersecurity Schema Framework (OCSF) events, categories, and profiles using Media over QUIC Transport (MOQT) tracks. The mapping leverages MOQT's hierarchical data model and publish/subscribe architecture to enable real-time, in-network cached, scalable distribution of cybersecurity event data across networks while maintaining OCSF's structured taxonomy and semantic relationships.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 April 2026.

Copyright Notice

Table of Contents

1. Introduction

The Open Cybersecurity Schema Framework (OCSF) [OCSF] provides a standardized approach to representing cybersecurity event data through a well-defined taxonomy of event classes, categories, and profiles. As cybersecurity operations increasingly require real-time threat intelligence sharing and event correlation across distributed systems, there is a need for an efficient transport mechanism that can scale to handle high-volume event streams while preserving the semantic structure of OCSF data.

Named data networking defines a computing and communication paradigm where bits being distributed are assigned network unique names. This is akin to HTTP's semantics of request, for a named data object, and response, containing the requested object. This paradigm enables a publish/subscribe based data delivery, where the consumers subscribe to interested data (via their names), publisher(s) produce named data that are delivered to matching subscribers, over a network that is capable of distributing and caching the named data. Media over QUIC Transport (MOQT) [MOQT] is an example of one such protocol that defines a publish/subscribe protocol optimized for low-latency, high-scale content distribution. MOQT's hierarchical data model of tracks, groups, and objects provides natural alignment opportunities with OCSF's structured taxonomy, while its publish/subscribe architecture enables efficient distribution of cybersecurity events to multiple consumers via in-network caches called MOQ Relays.

This document defines a comprehensive mapping specification that enables OCSF events to be efficiently distributed using MOQT transport.

1.1. Motivation

Current cybersecurity event distribution architectures often rely on traditional message queuing systems or HTTP-based APIs that may not adequately address the unique requirements of real-time security operations:

* Latency Sensitivity: Security events often require sub-second distribution to enable timely threat response and correlation.

* Scale Requirements: Modern security infrastructures generate millions of events per second that must be efficiently distributed to multiple consuming systems.

* Selective Consumption: Different security tools and analysts require different subsets of the overall event stream, necessitating efficient filtering capabilities.

* Distributed Caching: Security event streams benefit from in-network caching at relay nodes to reduce latency for geographically distributed consumers, enable efficient historical event retrieval, and provide redundancy for critical security data during network partitions or publisher failures.

MOQT addresses these requirements through its design for low-latency, high-scale content distribution with fine-grained subscription capabilities. The mapping defined in this document enables OCSF implementations to leverage these capabilities while maintaining full compatibility with the OCSF specification. The distributed relay architecture provides intelligent caching and geographic distribution, enabling efficient access to both real-time and historical security data while reducing bandwidth requirements and improving resilience.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3. Terminology

This document uses the following terms in addition to those defined in [MOQT] and [OCSF]:

OCSF Publisher: An entity that produces OCSF events and publishes them to MOQT tracks.

OCSF Subscriber: An entity that subscribes to MOQT tracks containing OCSF events.

Category Track: A MOQT track that carries events from a specific OCSF category.

Event Class Track: A MOQT track that carries events from a specific OCSF event class.

Profile Track: A MOQT track that carries events filtered by a specific OCSF profile.

2. OCSF Data Model Overview

This section provides a brief overview of the OCSF data model elements that are relevant to the MOQT mapping. For complete details of the OCSF specification, refer to [OCSF].

2.1. Event Classes

OCSF event classes serve as the fundamental building blocks of cybersecurity events, providing structured templates that define specific types of security-relevant activities. Each event class establishes a standardized format with well-defined attribute sets and semantic meanings, enabling consistent representation of security events across different systems and organizations.

The OCSF specification includes numerous event classes covering diverse security domains including system operations (File System Activity, Process Activity), network communications (Network Activity, HTTP Activity), security detections (Security Finding, VulnerabilityFinding), identity management (Authentication, Authorization), and application behaviors (API Activity, Web Resources Activity).

2.2. Categories

Categories represent the highest level of organization within the OCSF taxonomy, providing logical groupings that reflect the operational domains of cybersecurity activities. Each category receives a unique category_uid identifier and encompasses multiple related event classes that share common characteristics, operational contexts, or analytical purposes. Categories enable security practitioners to focus on specific domains of activity while maintaining the flexibility to correlate events across different categories when necessary

The framework defines following fundamental categories that cover the primary domains of cybersecurity activity:

System Activity (category_uid: 1): Encompasses operating system and host-based activities including file system operations, process management, and registry modifications. This category contains event classes such as File System Activity, Process Activity, and Registry Activity that capture the fundamental behaviors of operating systems and applications running on endpoints.

Findings (category_uid: 2): Contains security discoveries and threat detections generated by security tools and analysis systems. This category includes Security Finding events for malware

detections, Vulnerability Finding events for system weaknesses, and Compliance Finding events for policy violations. These events represent the output of security monitoring and assessment activities.

Identity & Access Management (category_uid: 3): Covers user authentication, authorization, and identity lifecycle activities. This category includes Authentication events for login attempts, Authorization events for access control decisions, and Account Change events for user provisioning activities. These events are essential for identity governance and access management systems.

Network Activity (category_uid: 4): Describes network communications and traffic patterns including connection events, protocol-specific activities, and network security controls. This category encompasses Network Activity events for general network connections, HTTP Activity events for web traffic, and DNS Activity events for domain name resolution activities.

Discovery (category_uid: 5) includes network scanning, service enumeration, reconnaissance activities, and asset discovery events. This category captures both legitimate discovery activities and potentially malicious reconnaissance behaviors.

Application Activity (category_uid: 6) captures application-specific events, API calls, software behavior, and application security events. This category encompasses the diverse behaviors of applications, services, and software components.

2.3. Profiles

OCSF profiles are sophisticated framework constructs that provide a cross-cutting mechanism to augment event classes and objects with specialized attribute sets that describe specific aspects of activities and findings. Rather than creating an explosion of specialized event classes for every possible combination of attributes, profiles provide an elegant "mix-in" approach that allows reuse of fundamental class semantics while adding contextual information relevant to specific operational environments, technologies, or analytical perspectives. Profiles enable the framework to remain both comprehensive and maintainable by separating core event semantics from contextual augmentations.

Core OCSF Profiles:

* *Security Control Profile*: Augments events with MITRE ATT&CK technique mappings, malware classifications, disposition information, and security control effectiveness data. This profile is crucial for threat intelligence and security analytics applications.

* *Host Profile*: Adds device and actor context information for host-based activities, including detailed system information, user context, and host-specific identifiers. This profile enables correlation of activities across host-based security tools.

* *Cloud Profile*: Incorporates cloud platform-specific attributes including cloud_partition, region, availability zones, and cloud service identifiers. This profile supports cloud-native security monitoring and compliance.

* *Container Profile*: Adds container orchestration attributes including container names, images, orchestration platforms, and container runtime information. This profile enables security monitoring in containerized environments.

* *Date/Time Profile*: Provides human-readable datetime counterparts to timestamp fields, supporting systems that require both machine-readable timestamps and human-friendly time representations.

* *Network Proxy Profile*: Adds network proxy-specific attributes for events that traverse proxy infrastructure, including proxy identifiers, policies, and routing information.

3. MOQT Data Model Overview

This section summarizes the MOQT data model elements used in the mapping. For complete details, refer to [MOQT].

3.1. Tracks

MOQT tracks represent stream of temporally ordered data and serve as the primary unit of subscription and organization in the MOQT architecture. In the context of OCSF event distribution, tracks provide logical channels through which related cybersecurity events flow from publishers to subscribers.

Using an hierarchical namespace for Track naming allows for organizing OCSF events by category, class, profile, or other taxonomic dimensions, enabling subscribers to precisely target the event types they need without receiving unnecessary data.

3.2. Groups

MOQT groups provide the primary mechanism for temporal organization and subscription synchronization in MOQT and can provide the fundamental boundary for event batching in OCSF distributions In OCSF applications, groups are typically organized around time windows (e.g., events from the same second or minute) or logical boundaries (e.g., events from the same security incident), enabling efficient batch processing and maintaining temporal relationships between related security events.

3.3. Objects

MOQT objects are the fundamental data units within groups and serve as the actual containers for OCSF event data. Each object carries one or more cybersecurity events in their payload, with the object structure designed to optimize both individual event access and batch processing scenarios.

Objects can be organized into subgroups for dependency management and priority handling. In OCSF implementations, objects may contain single high-priority events (such as critical security findings) for immediate processing, or multiple related events batched together for efficient bandwidth utilization. The immutable nature of objects ensures that security event data maintains integrity throughout the distribution process, while the caching capabilities enable efficient replay and historical analysis of security events.

4. OCSF to MOQT Mapping Strategy

4.1. Mapping Approach

The mapping strategy proposes using an hierarchical structure to represent OCSF’s taxonomy while maintaining efficient distribution characteristics. The approach uses multiple complementary track organizations:

1. *Category-based tracks*: Organize events by OCSF category for domain-specific consumption

2. *Event class tracks*: Provide fine-grained subscriptions to specific event types

3. *Profile-filtered tracks*: Enable cross-category subscriptions based on profile attributes

4. *Composite tracks*: Support complex subscription patterns combining multiple criteria

This multi-dimensional approach enables subscribers to choose the most appropriate granularity for their use case while maintaining efficient publisher operation.

4.2. Category Tracks

Category tracks organize events by OCSF category, providing domain-specific event streams.

The track namespace structure uses a hierarchical organization with the following tuples:

("ocsf", {version}, {organization}, {deployment}, "category", {category_level}, {endpoint_id})

Where:

* *version*: OCSF schema version (e.g., "1.0.0")

* *organization*: Deploying organization identifier

* *deployment*: Specific deployment or environment identifier

* *category_level*: OCSF category identifier as defined by category_uid (e.g., "1" for System Activity)

* *endpoint_id*: Unique endpoint identifier for the publishing device/system.

Track names within the namespace follow a single tuple structure that combines the track type and identifier:

"activity/{type_uid}"

where type_uid is OCSF event/finding type ID. It identifies the event’s semantics and structure. The value is calculated by as: class_uid * 100 + activity_id.

Each category track:

* Contains all events from event classes within the category

* Uses temporal grouping based on configurable time windows

* Maintains event ordering within groups by timestamp

* Supports efficient category-level subscriptions

Group organization within category tracks uses time-based windows:

* Group ID corresponds to time window (e.g., seconds since epoch)

* All events within the time window belong to the same group

* Window size is configurable based on volume and latency requirements

* Late-arriving events may create out-of-order groups

Object organization within groups orders events by the OCSF timestamp (time attribute), then by Event class UID. ObjectID of the first object correspond to the first event in the group, and subsequent objects are assigned sequentially.

4.3. Event Class Tracks

Event class tracks provide fine-grained subscriptions to specific event types.

The track namespace structure uses a hierarchical organization with the following tuples:

("ocsf", {version}, {organization}, {deployment}, "events", {event_class}, {endpoint_id})

Where:

* *version*: OCSF schema version (e.g., "1.0.0")

* *organization*: Deploying organization identifier

* *deployment*: Specific deployment or environment identifier

* *event_class*: OCSF Event Class UID as defined by class_uid (e.g., "1001" for File System Activity)

* *endpoint_id*: Unique endpoint identifier for the publishing device/system.

Track names within the namespace follow a single tuple structure that combines the track type and identifier:

"activity/{activity_uid}"

where activity_uid is the OCSF activity ID.

Each event class track:

* Contains events from a single OCSF event class

* May optionally filter by specific activity_id values

* Uses smaller temporal windows due to focused scope

* Enables efficient processing of specific event types

TODO: Define Group and Object organization within event class tracks.

4.4. Profile-Filtered Tracks

Profile tracks contain events that include specific OCSF profile attributes, enabling cross-category subscriptions based on profile criteria.

The track namespace structure uses a hierarchical organization with the following tuples:

("ocsf", {version}, {organization}, {deployment}, "profile", {profile_name}, {endpoint_id})

Where:

* *version*: OCSF schema version (e.g., "1.0.0")

* *organization*: Deploying organization identifier

* *deployment*: Specific deployment or environment identifier

* *profile_name*: OCSF Profile Name as defined by profile_name (e.g., "security" for Security Control profile)

* *endpoint_id*: Unique endpoint identifier for the publishing device/system.

Track names within the profile namespace has the following structure:

{filter_criteria}

where filter_criteria can include:

* Presence of specific profile attributes

* Profile attribute value ranges or enumeration matches

* Disposition values for Security Control profile

* MITRE ATT&CK technique classifications

Few examples of profile track name filter criteria include:

```
disposition/blocked  # Security Control profile with blocked disposition
attack/T1055         # Security Control profile with MITRE ATT&CK technique T1055
device/windows       # Host profile for Windows devices
partition/aws        # Cloud profile for AWS partitions
```

5. Event Mapping to Objects

5.1. Object Structure

OCSF events are mapped to MOQT objects using a structured approach that preserves all event information while optimizing for transport efficiency.

Object payload structure:

```
OCSF-Object {
  Header (
    Format Version (8),
    Compression Type (8),
    Event Count (i),
    Batch Metadata (...)
  ),
  Events [
    Event Length (i),
    Event Data (...)
  ] ...
}
```

Where:

* *Format Version*: Version of the OCSF-MOQT mapping format

* *Compression Type*: Optional compression algorithm identifier

* *Event Count*: Number of events in this object

* *Batch Metadata*: Optional metadata about the event batch

* *Event Length*: Length of individual event data

* *Event Data*: Serialized OCSF event (JSON, MessagePack, etc.)

Single-event objects (Event Count = 1) are used for:

* High-priority events requiring immediate distribution

* Large events that would cause batching delays

* Events with specific delivery requirements

Multi-event objects (Event Count > 1) are used for:

* High-volume event streams requiring efficiency

* Events that can tolerate small batching delays

* Bandwidth-constrained environments

5.2. Metadata Mapping

MOQT object metadata carries essential OCSF event information for relay processing and subscription filtering. Object priority is derived from OCSF severity and disposition attributes.

5.3. Temporal Grouping

OCSF events are grouped into MOQT groups based on temporal windows that balance latency and efficiency requirements. An implementation MAY choose to group events based on time intervals, event counts, or a combination of both. Selection of the grouping strategy depends on the specific application requirements and the characteristics of the event stream.

Below are few considerations for selecting the grouping strategy:

Window sizing strategies, where a fixed-interval windows when employed can provide predictable group boundaries. The Group ID can be calculated as floor(event.time / window_size_ms), for examples. Adaptive window sizing strategies can be used to adjust the window size based on event volume, allowing for smaller windows during low-volume periods and larger windows during high-volume periods. This can help balance latency and efficiency.

Implementations may choose to use event count windows, where groups size is limited to a fixed number of events. This ensures consistent group sizes, and may result in variable time spans depending on event frequency.

Hybrid approaches that combine time and count limits can also be used, where groups are formed based on a maximum time interval or a maximum number of events, whichever comes first. This allows for flexibility in handling varying event rates while maintaining efficient grouping.

Recommended window configurations:

* High-frequency categories (System Activity): 1-5 second windows

* Medium-frequency categories (Network Activity): 5-30 second windows

* Low-frequency categories (Findings): 30-300 second windows

* Critical events: Immediate group closure (sub-second)

6. Examples

6.1. Subscription Patterns

This section provides examples for common subscription pattern defined in this specficaition

6.1.1. Category Subscriptions

Basic category subscription for Application Activity Category (6) and API Activity Class (6003) for delete activity (activity_id = 4), whose type_uid is 600304:

```
SUBSCRIBE {
  Request ID: 42,
  Track Namespace: ("ocsf", "1.0.0", "acme", "prod", "category", "4", "alice001"),
  Track Name: "activity/600304",
  Start Group: Latest,
  End Group: None,
  Parameters: {}
}
```

Time-bounded category subscription for event-log activity by a server endpoint under System Activity Category (1):

```
SUBSCRIBE {
  Request ID: 43,
  Track Namespace: ("ocsf", "1.0.0", "acme", "prod", "category", "1", "server-112-7777"),
  Track Name: "activity/1008/6",
  Start Group: 1640995200,  # Jan 1, 2022 00:00:00 UTC
  End Group: 1640998800,    # Jan 1, 2022 01:00:00 UTC
  Parameters: {}
}
```

6.1.2. Event Class Subscriptions

Event class subscriptions provide focused consumption of specific event types:

Event class subscription for decryption operation on a file under File System Activity (1001):

```
SUBSCRIBE {
  Request ID: 45,
  Track Namespace: ("ocsf", "1.0.0", "acme", "prod", "events", "1001", "ws-alice-01"),
  Track Name: "activity/11",
  Start Group: Latest,
  End Group: None,
  Parameters: {
    Group Order: Ascending
  }
}
```

Historical event class subscription (using FETCH) (Authentication events for revoking privileges):

```
FETCH {
  Request ID: 47,
  Track Namespace: ("ocsf", "1.0.0", "acme", "prod", "events", "3001", "server-112-7777")
,
  Track Name: "activity/2",
  Start Group: 1640995200,  # Historical start
  End Group: 1640998800,    # Historical end
  Parameters: {
    Group Order: Ascending,
    Priority: 64
  }
}
```

6.1.3. Profile-Based Subscriptions

Filtered profile subscription (Blocked events only):

```
SUBSCRIBE {
  Request ID: 49,
  Track Namespace: ("ocsf", "1.0.0", "acme", "prod", "profile", "security", "api-gateway-
01"),
  Track Name: "disposition/blocked",
  Start Group: Latest,
  End Group: None,
  Parameters: {}
}
```

Host profile subscription to learn about devices that are marked as high risk levels. (for host-based activities):

```
SUBSCRIBE {
  Request ID: 50,
  Track Namespace: ("ocsf", "1.0.0", "acme", "prod", "profile", "host", "network-device-0
1"),
      Track Name: "risk_level/4",
  Start Group: Latest,
  End Group: None,
  Parameters: {}
}
```

This will receive both System Activity events (native profile) and Network Activity events (augmentation profile) that include host context.

6.2. OCSF Event Publishing Examples

This section provides examples of publishing OCSF events to MOQT tracks using the PUBLISH message.

6.2.1. Publishing to Category Tracks

Publishing a File System Activity event to System Activity category, event activity of type "Open" with activity_id 1:

```
PUBLISH {
  Request ID: 100,
  Track Namespace: ("ocsf", "1.0.0", "acme", "prod", "category", "1", "ws-alice-01"),
  Track Name: "activity/1",
  Parameters: {}
}
```

Object data shall contain the OCSF event:

Object Header:

```
Track Alias: 1
Group ID: 1640995260     # Current time window
Object ID: 1
Object Send Order: 1
Object Status: 0 (Normal)
```

Object Payload (JSON example):

```
   {
     "activity_id": 1,
     "activity_name": "Open",
     "category_name": "System Activity",
     "category_uid": 1,
     "class_name": "File System Activity",
     "class_uid": 1001,
     "device": {
       "hostname": "workstation-01",
       "os": {"name": "Windows", "version": "10"}
     },
     "file": {
       "name": "document.pdf",
       "path": "C:\\Users\\alice\\Documents\\document.pdf",
       "size": 2048576
     },
     "metadata": {
       "version": "1.0.0",
       "uid": "550e8400-e29b-41d4-a716-446655440000"
     },
     "severity": "Informational",
     "severity_id": 1,
     "time": 1640995260123,
     "type_name": "File System Activity: Open",
     "type_uid": 100101
   }
```


6.2.2. Publishing to Profile Tracks

Publishing a security event with Security Control profile (using augmentation approach):

```
PUBLISH {
  Request ID: 102,
  Track Namespace: ("ocsf", "1.0.0", "acme", "prod", "profile", "scurity", "endpoint-secu
rity-01"),
  Track Name: "disposition/blocked",
  Parameters: {}
}
```

Object data shall contain the OCSF event:

Object Header:

```
Track Alias: 3
Group ID: 1640995260
Object ID: 1
Object Send Order: 1
Object Status: 0
```

Object Payload (includes Security Control profile attributes):


```
{
  "activity_id": 1,
  "activity_name": "Create",
  "category_name": "System Activity",
  "category_uid": 1,
  "class_name": "Process Activity",
  "class_uid": 1007,
  "device": {
    "hostname": "endpoint-01",
    "os": {"name": "Windows", "version": "10"}
  },
  "process": {
    "name": "malware.exe",
    "pid": 1234,
    "file": {
      "name": "malware.exe",
      "hashes": [
        {
          "algorithm": "SHA-256",
          "value":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
        }
      ]
    }
  },
  "disposition": "Blocked",
  "disposition_id": 2,
  "malware": [
    {
      "name": "Trojan.Generic",
      "classification_ids": [2]
    }
  ],
  "attacks": [
    {
      "technique": {
        "name": "Process Injection",
        "uid": "T1055"
      }
    }
  ],
  "metadata": {
    "version": "1.0.0",
    "uid": "550e8400-e29b-41d4-a716-446655440002",
    "profiles": ["security"]
  },
```

```
    "severity": "High",
    "severity_id": 4,
    "time": 1640995260789,
    "type_name": "Process Activity: Create",
    "type_uid": 100701
  }
```


7. Relay Considerations

TODO: Add considerations for MOQ Relays in the context of OCSF-MOQT mapping.

8. Security Considerations

TODO: Define security considerations for the OCSF-MOQT mapping.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[MOQT] Nandakumar, S., Vasiliev, V., Swett, I., and A. Frindell, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-12, 23 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-12>.

9.2. Informative References

[OCSF] Agbabian, P., "Understanding the Open Cybersecurity Schema Framework", Version 1.16, September 2024.

Appendix A. References

A.1. Normative References

[MOQT] "Media over QUIC Transport (MOQT)", draft-ietf-moq-transport (work in progress)

[OCSF] "Open Cybersecurity Schema Framework (OCSF)", version 1.0.0,

https://schema.ocsf.io/

Appendix B. Acknowledgments

Appendix C. Contributors

Authors' Addresses

Suhas Nandakumar
Cisco Systems, Inc.
Email: snandaku@cisco.com

Cullen Jennings
Cisco Systems, Inc.
Email: fluffy@iii.ca