Open Command and Control (OpenC2) Language Description Document

Version 1.0 – Release Candidate 3
17 February 2017

FOREWORD

The Open Command and Control Forum (OpenC2 or the Forum) supports the cyber defense community of interest by developing and promoting the adoption of the OpenC2 language, data models, prototype implementations, and reference material that addresses the command and control of cyber defense components, technologies, and systems.

This Forum serves developers, users, and the entire cybersecurity ecosystem by providing a set of shared resources to expand the use of standardized command and control for cyber defense activities, to enable technology vendors building orchestration and cyber response technologies, and to assist developers in producing response technologies that can be readily used in coordinated responses. The goal of the Forum is to provide and open and collaborative environment and to present its findings and artifacts to recognized standards bodies for the standardization of the command and control language.

This document represents the outcome of collaboration between technology vendors, government agencies, and academia on the topic of command and control for cyber defensive measures. We gratefully acknowledge their contributions to the definition of the OpenC2 language. As we exercise the language in reference implementations and in real-world operations, we expect to continue to refine the language to ensure its suitability to support machine-to-machine command and control communications in response to cyber threats in cyber-relevant time.

Visit openc2.org for other on-line resources.

TABLE OF CONTENTS

1. Introduction

Cyberattacks are increasingly more sophisticated, less expensive to execute, dynamic, and automated. Current cyber defense products are typically integrated in a unique or proprietary manner and statically configured. As a result, upgrading or otherwise modifying tightly integrated, proprietary cyber defense's functional blocks is resource intensive; cannot be realized within a cyber-relevant timeframe; and the upgrades may degrade the overall performance of the system.

Future cyber defenses against current and pending attacks require the integration of new or upgraded functional capabilities, the coordination of responses across domains, synchronization of response mechanisms, and deployment of automated actions in cyber relevant time.

Standardization of the languages, including lexicons, syntaxes, and encodings, used within the interfaces and protocols necessary for machine-to-machine command and control communications in cyber relevant time will enable cyber defense system flexibility, interoperability, and responsiveness in cyber relevant time.

1.1 Purpose

The purpose of the Open Command and Control (OpenC2) Language Description Document is to define a lexicon language and semantics at a level of abstraction that will enable the coordination and execution of command and control of cyber defense components between and within networks. It is expected that the OpenC2 language will define profiles (i.e.., applicable commands, applicable values) by community groups for specific cyber defense functions such as Software Defined Networking, Firewall, routing.

1.2 Scope

The scope of this document is to create a lexicon of actions and define the semantics, syntax and other aspects of a language that will couple an action with the target of the actions, and the entities that execute the actions. The document also defines an extensible syntax to accommodate attributes that further specify the targets, and modify actions to support a wide range of operational environments.

Other aspects of OpenC2, such as implementation considerations, further refinement of the lexicon to accommodate specific cyber defense functions, encoding of commands for machine to machine communications and reference implementations will be addressed in other artifacts. These other efforts will be consistent with this language description.

The definition of a language such as OpenC2 is necessary but insufficient to enable future cyber defenses. OpenC2 commands can be carried within any number of constructs (e.g., STIX, workflows, playbooks, API's). In addition, OpenC2 is designed to be flexible, agnostic of external protocols that provide services such as transport, authentication, key management and other services. Cyber defense implementations must consider and will require other protocols and security services.

1.3 Intended Audience

This OpenC2 Language Description Document is intended for organizations investigating the implementation of automated pre-approved cyber defensive measures as well as academia and industry partners involved with the development and integration of security orchestration, network components or services, endpoint security applications, and security services for cyber defenses.

1.4 Document Overview

Section 1, Introduction, describes the impetus for the OpenC2 language and lays out the purpose, scope, and intended audience of the document.

Section 2, Background, describes the design principles for the language and how the language can be contextualized for different operating environments.

Section 3, OpenC2 Language, describes the abstract syntax and the basic building blocks of the language. It also further specifies the vocabulary for actions, universal modifiers, action specific modifiers and a default namespace for targets and target specifiers..

Section 4, Example OpenC2 Usage, provides examples of OpenC2 command constructs. For each action, the supported targets, actuators, and action specific modifiers are identified and example usages are provided.

Section 5, Example OpenC2 Use Case, depicts an example use case for mitigating an evil domain. The use case shows the OpenC2 commands that could be used to mitigate the attacks or vulnerabilities and where they could be applied.

Appendix A, Example OpenC2 Commands, contains example OpenC2 commands organized in tables by OpenC2 action. These example commands were based on use cases provided by government agencies, critical infrastructure, industry (e.g., security orchestrator, actuator, and sensor) and academia.

2. Background

2.1 Design Principles

OpenC2 can be implemented in a variety of systems to perform the secure delivery and management of command and control messages in a context-specific way. OpenC2 commands are vendor neutral and message fabric agnostic, thus can be incorporated in different architectures and environments (such as connection oriented, connectionless, pub-sub, hub and spoke, etc.).

OpenC2 was designed to have a concise set of commands are extensible in order to provide context specific details. Conciseness ensures minimal overhead to meet possible latency and overhead constraints while extensions enable greater utility and flexibility.

There is an underlying assumption that issuing OpenC2 commands are event-driven and that an action is warranted. OpenC2 was designed to focus on the actions that are to be executed in order to thwart an attack, mitigate some vulnerability or otherwise address a threat. The exchange of indicators, rationale for the decision to act and/or threat information sharing are beyond the scope of OpenC2 and left to other standards such as STIX, TAXII etc.

The actual performance and efficacy of OpenC2 will be implementation-specific and will require the incorporation of other technologies. The OpenC2 design principles include the following:

- Support cyber relevant response time for coordination and response actions.
- Be infrastructure, architecture, and vendor agnostic.
- Support multiple levels of abstraction, necessary to permit the contextualization of commands for a wide variety of operating environments.
- Permit commands to be invoked that are either tasking/response actions or notifications.
    - Tasking/response actions result in a state change.
    - Notifications require supporting analytics/decision processes.
- Provide an extensible syntax to accommodate different types of actions, targets, and actuators (e.g., sensor, endpoint, network device, human) at varying levels of specificity. .
- Ensure the OpenC2 command is independent of a message construct that provides transport, identifies priority/ quality of service, and supports security attributes.

By design, OpenC2 is dependent upon but agnostic of the transport infrastructure and message fabric. Confidentiality, integrity, availability and authentication must be identified and provisioned by the message fabric.

Traditional command and control implementations utilize complete, self-standing constructs. OpenC2 decouples the actions from the targets of the actions and from the recipients of the commands. An OpenC2 command is not complete until an action is paired with a target, providing the command context for the action. This enables the OpenC2 language to be more concise, yet still support the entire C2 space. This characteristic of OpenC2 also permits a more flexible and extensible approach to accommodate future technologies and varying network environments.

2.2 OpenC2 and Deployment Environments

OpenC2 is defined at a level of abstraction such that an inter-domain tasking or coordination effort can be described without requiring in depth knowledge of the recipient network's components, but through the use of specifiers and modifiers, enough detail can be appended to carry out specific tasks on particular devices to support intra-domain command and control.

This level of abstraction permits end to end applicability of OpenC2. As depicted in Figure 2-1, an OpenC2 command is sent to enable coordination or send a high level tasking from the peer or upper tier enclave. An OpenC2 command received by an enclave will trigger events within the enclave to annotate the command with context specific information so that specific devices within the enclave can respond appropriately. This allows the enclave to take advantage of this context-specific knowledge to interpret and appropriately execute OpenC2 commands .

Each network contextualizes an OpenC2 action for the specific sensors and actuators within its environment so it can further specify the command to reflect the implementations of which it is capable. Context-specific modifiers provide an ability to further specify the action while enabling the set of actions to remain tightly constrained. This minimizes the overhead, permits further contextualization of the OpenC2 commands for specific environments, and thereby enables flexibility and extensibility.

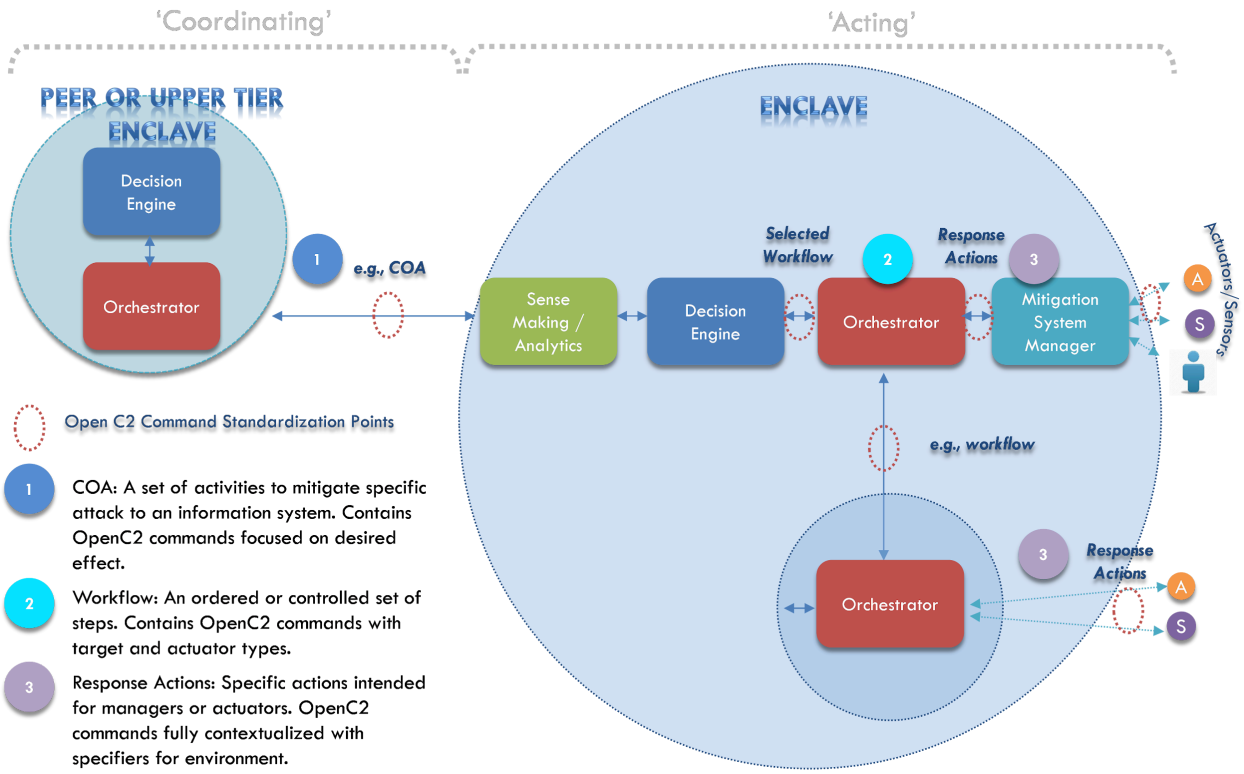

Figure 2-1. OpenC2 Deployment Environments

For example, an organization may have executed a series of actions to protect against a particular attack that was signaled by an external indicator (such as a STIX message). In order to elicit a consistent response across an organization (whether hierarchical or peer to peer), a complex course of action can be constructed and shared. The use of standardized OpenC2 commands will be more precise and more quickly actionable than a set of recommended steps within a text document (e.g., flash), which must be parsed, analyzed, and interpreted, prior to execution. Standardizing OpenC2 commands helps to ensure a more uniform response at enterprises/enclaves that reflects enterprise-wide level decisions.

3. OpenC2 Language

3.1 Overview

The OpenC2 language is designed at a level of abstraction high enough such that it enables persistence as technologies advance and is implementation agnostic, but enough precision so that the need for specifiers and modifiers is limited.

3.2 Abstract Syntax

Conceptually, an OpenC2 command has the following form:

```
(
      ACTION = <ACTION_TYPE>,
      TARGET (
            type = <data-model>:<TARGET_TYPE>,
            <target-specifier>
      ),
      ACTUATOR (
            type = <data-model>:<ACTUATOR_TYPE>,
            <actuator-specifier>
      ),
      MODIFIERS (
            <list-of-modifiers>
      )
)
```

Fields denoted with angle brackets ("<>") are replaced with the appropriate details. Some of the fields are considered optional. The table below describes these fields semantically and whether they are required, optional or ignored in certain situations. Actual encoding will leverage pre-existing conventions and notations such as XML, JSON, TLV or others. .

The following table contains the description of the fields that can be contained in an OpenC2 command.

Table 3-1. OpenC2 Command Field Descriptions

Field	Description
ACTION	Required. The task or activity to be performed (i.e., the 'verb').
data-model	Required. The data model for the TARGET.
TARGET	Required. The object of the action. The ACTION is performed on the TARGET.
type	Required. The TARGET type will be defined within the context of a namespace.
target-specifier	Optional. The specifier further describes a specific target, a list of targets, or a class of targets.
ACTUATOR	Optional. The subject of the action. The ACTUATOR executes the ACTION on the TARGET.
type	Required if the actuator is included, otherwise not applicable. The ACTUATOR type will be defined within the context of a namespace.
data-model	Required if the actuator is included, otherwise not applicable. The data model for the ACTUATOR.
actuator-specifier	Optional if the actuator is included, otherwise not applicable. The specifier further describes a specific actuator, a list of actuators, or a class of actuators.
MODIFIERS (<list-of-modifiers>)	Optional. Provide additional information about the action such as date/time, periodicity, duration, and location.

There are cases where an ACTION and TARGET are sufficient to complete the command, especially in the case of inter-domain commands where the method or approach to complete or execute the action can be determined within the receiving domain/enclave.

The majority of commands within an enclave will have an ACTION, TARGET and ACTUATOR. Inclusion of the ACTUATOR provides additional context for the command as a whole and enables precision. .

Specifiers for TARGETs and ACTUATORs are optional and can be used to provide context specific information that could be used to reflect the local environment, policies, and operational conditions within an enterprise/enclave. Specifiers can call out a specific target/actuator, a list of targets/actuators, or a class of targets/actuators.

Modifiers to the ACTION are optional and are used to provide effect based context to the ACTION. Modifiers are further discussed in Section 3.2.5.

Table 3-2 illustrates the use of specifiers and modifiers to extend the range of OpenC2 commands to cover the higher level 'strategic' commands to the unambiguous enclave-specific use case. This provides greater flexibility to the language and allows the OpenC2 actions to be further contextualized for the mission environment. The table below provides some examples of the different levels of specificity achievable in an OpenC2 command.

Table 3-2. OpenC2 Syntax Flexibility Examples

Description	Action	Target	Actuator	Modifier
		Target-Specifier	Actuator-Specifier	
Block traffic to/from specific IP address(es) [effects-based, no actuator specified]; suitable for inter-domain coordination	DENY	Network Connection		
		Source and/or Destination IP Address(es)		
Block traffic at all network devices [specify actuator class]; suitable for inter-domain coordination or as a command to an orchestration engine which further contextualizes to the enclave's environment	DENY	Network Connection	Network (any devices)	
		Source and/or Destination IP Address(es)		
Block traffic at network routers [specify type of	DENY	Network Connection	Network.router	

network device actuator]; suitable within an enclave		Source and/or Destination IP Address	(optional)	
Block traffic at specific network router; [specify identity of network router]; suitable within an enclave	DENY	Network Connection	Network.router	
		Source and/or Destination IP Address	Router identity	
Block access to bad external IP by null routing; [specify method of performing action]; suitable within an enclave	DENY	Network Connection	Network.router	Method= blackhole
		Source and/or Destination IP Address	(optional)	

3.2.1 Action

All OpenC2 commands start with an ACTION which indicates the type of command to perform such as gather and convey information, control activities and devices, and control permissions and access. The range of options and potential impact on the information system associated with a particular ACTION is a function of the ACTUATOR. For cases that involve multiple options for an ACTION, modifiers may be used.

Refer to Section 3.3 for the list of ACTIONs and their definitions and usage.

3.2.2 Target

All OpenC2 commands include a TARGET. The TARGET is the object of the ACTION (or alternatively, the ACTION is performed on the TARGET). Targets include objects such as network connections, URLs, hashes, IP addresses, files, processes, fully qualified domain names etc. .

3.2.3 Actuator

An ACTUATOR[1] is the entity that puts command and control into motion or action. The ACTUATOR is the subject of the ACTION which performs the ACTION on the TARGET. There are varying levels of abstraction and functionality for an ACTUATOR ranging from a specific sensor to an entire system or even system of systems.

The source of a command may need to communicate an action that must be taken against a target, but will not necessarily have knowledge of the cyber defense technologies deployed in other enclaves so the inclusion of an actuator is optional within an OpenC2 command. As a command is propagated through the system and context specific information is gained, the command can appended with an actuator and appropriate specifiers.

There will be only one ACTUATOR type per OpenC2 command. The actuator namespace is specified in the OpenC2 profiles. .

3.2.4 Specifiers

"Specifiers" are used to identify specific individual or groups of targets or actuators. Table 3-3 illustrates how the commands are appended with specifiers as context specific details become available. The actuator specifiers presented in Table 3-3 are for illustrative purposes. The actual specifiers are defined in the appropriate actuator profiles.

Table 3-3. Example Usage of Specifiers

Description	Action	Target	Actuator	Modifier
		Target-Specifier	Actuator-Specifier	
Block malicious URL	DENY	URI/URL		
		Value Condition = Equals		
Quarantine Artifact with particular byte string	QUARANTINE	Artifact		
		Condition = Contains		

[1] Some academic circles model all cyber defense components as sensors and/or actuators. It is acknowledged that OpenC2 will be used for C2 of sensors as well, but in the interest of being concise within this document, actuators encompass sensors.

Block access to external IP address by null routing at specific network routers	DENY	Network Connection	Network router	
		Condition = Contains	Manufacturer, Model, Serial Number Value = 123	

3.2.5 Modifiers

“Modifiers” provide additional precision about the action such as time, periodicity, duration, or other details on what is to be done. Modifiers can denote the when, where, and how aspects of an action. The modifier can also be used to convey the need for acknowledgement or additional status information about the execution of an action. Modifiers are similar to specifiers in that they can provide additional context specific details, and are intended to provide additional details for action/actuator pairs. A modifier may be “actuator-specific”, “action-specific”, or “universal” depending on the applicability of the modifier within the language.

Actuator-specific modifiers are described in Actuator Profiles. Action-specific are described in Section 4. Universal modifiers are described in the following table.

Table 3-4. Example Usage of Modifiers

Description	Action	Target	Actuator	Modifier
		Target-Specifier	Actuator-Specifier	
Shutdown a system, immediate	STOP	Device	endpoint	method = immediate
		Device Object Type	(optional)	
Start Process with Delay	START	Process	endpoint	Delay = duration
		Process Object Type	(optional)	
Quarantine a device	CONTAIN	Device	network	where (network
		Device Object Type	(optional)	

				segment, vlan)
Block access to suspicious external IP address by redirecting external DNS queries to an internal DNS server	DENY	Network Connection	DNS Server	method = sinkhole
		Network Connection Object Type		

3.3 Actions

This section defines the set of OpenC2 actions grouped by their general activity. The following table summarizes the definition of the OpenC2 actions. Subsequent sections will identify the appropriate targets for each action and the appropriate actuators for the action target pair. Further details will be defined in the actuator profiles. .

- Actions that Control Information:
  These actions are used to gather information needed to determine the current state or enhance cyber situational awareness. These actions typically do not impact the state of the target and are normally not detectable by external observers.

- Actions that Control Permissions:
  These actions are used to control permissions and manage accesses.

- Actions that Control Activities/Devices:
  These actions are used to control the state or the activity of a system, a process, a connection, a host, or a device (e.g., endpoint, sensor, actuator). The actions are used to execute tasks, adjust configurations, set and update parameters, and modify attributes.

- Sensor-Related Actions:
  These actions are used to control the activities of a sensor in terms of how to collect and provide the sensor data.

- Effects-Based Actions:
  Effects-based actions are at a higher level of abstraction for purposes of communicating a desired impact rather than a command to execute specific tasks within an enclave. This level of abstraction enables coordinated actions between enclaves, while permitting a local enclave to optimize its workflow for its specific environment.

  Implementation of an effects-based action requires that the recipient enclave has a

decision making capability because an effects-based action permits multiple possible responses.

- Response and Alert:
  RESPONSE is used to provide data requested as a result of an action. The RESPONSE message will contain the requested data and have a reference to the action that initiated the response. ALERT is used to signal the occurrence of an event or error. It is an unsolicited message that does not reference a previously issued action.

Table 3-5. Summary of Action Definitions

Actions that Control Information	
SCAN	The SCAN action is the systematic examination of some aspect of the entity or its environment in order to obtain information.
LOCATE	The LOCATE action is used to find an object either physically, logically, functionally, or by organization. This action enables one to tell where in the system an event or trigger occurred.
QUERY	The QUERY action initiates a single request for information.
REPORT	The REPORT action tasks an entity to provide information to a designated recipient of the information.
NOTIFY	The NOTIFY action is used to set an entity's alerting preferences.
Actions that Control Permissions	
DENY	The DENY action is used to prevent a certain event or action from completion, such as preventing a flow from reaching a destination (e.g., block) or preventing access.
CONTAIN	The CONTAIN action stipulates the isolation of a file or process or entity such that it cannot modify or access assets or processes that support the business and/or operations of the enclave.
ALLOW	The ALLOW action permits the access to or execution of a target.

Actions that Control Activities/Devices	
START	The START action initiates a process, application, system or some other activity.
STOP	The STOP action halts a system or ends an activity.
RESTART	The RESTART action conducts a STOP of a system or an activity followed by a START of a system or an activity.
PAUSE	The PAUSE action ceases a system or activity while maintaining state.
RESUME	The RESUME action starts a system or activity from a paused state.
CANCEL	The CANCEL action invalidates a previously issued action.
SET	The SET action changes a value, configuration, or state of a managed entity within an IT system.
UPDATE	The UPDATE action instructs the component to retrieve, install, process, and operate in accordance with a software update, reconfiguration, or some other update.
MOVE	The MOVE action changes the location of a file, subnet, network, or, process.
REDIRECT	The REDIRECT action changes the flow of traffic to a particular destination other than its original intended destination.
DELETE	The DELETE action removes data and files.
SNAPSHOT	The SNAPSHOT action records and stores the state of a target at an instant in time.
DETONATE	The DETONATE action executes and observes the behavior of a target (e.g., file, hyperlink) in a manner that is isolated from assets that support the business or operations of the enclave.
RESTORE	The RESTORE action deletes and/or replaces files, settings, or attributes to return the system to an

	identical or similar known state.
SAVE	The SAVE action commits data or system state to memory.
THROTTLE	The THROTTLE action adjusts the throughput of a data flow.
DELAY	The DELAY action stops or holds up an activity or data transmittal.
SUBSTITUTE	The SUBSTITUTE action replaces all or part of the data, content or payload in the least detectable manner.
COPY	The COPY action duplicates a file or data flow.
SYNC	The SYNC action synchronizes a sensor or actuator with other system components.
Sensor-Related Actions	
DISTILL	The DISTILL action tasks the sensor to send a summary or abstraction of the sensing information instead of the raw data feed.
AUGMENT	The AUGMENT action tasks the sensor to do a level of preprocessing or sense making prior to sending the sensor data.
Effects-Based Actions	
INVESTIGATE	The INVESTIGATE action tasks the recipient enclave to aggregate and report information as it pertains to an anomaly.
MITIGATE	The MITIGATE action tasks the recipient enclave to circumvent the problem without necessarily eliminating the vulnerability or attack point. Mitigate implies that the impacts to the enclave's operations should be minimized while addressing the issue.
REMEDIATE	The REMEDIATE action tasks the recipient enclave to eliminate the vulnerability or attack point.

	Remediate implies that addressing the issue is paramount.
Response and Alert	
RESPONSE	RESPONSE is used to provide any data requested as a result of an action. RESPONSE can be used to signal the acknowledgement of an action, provide the status of an action along with additional information related to the requested action, or signal the completion of the action. The recipient of the RESPONSE can be the original requester of the action or to another recipient(s) designated in the modifier of the action.
ALERT	ALERT is used to signal the occurrence of an event.

3.4 Target Vocabulary

The TARGET is the object of the ACTION (or alternatively, the ACTION is performed on the TARGET). OpenC2 defines a default TARGET Data Model to support all of the actions. It is derived largely on the STIX Cyber Observables v2.x.

In addition to the default TARGET Data Model, the OpenC2 syntax can support any other data model. To differentiate alternative data models, a data model prefix is used to qualify the target type. The default target data model will prefix "openc2:" to the target type. The implementer will need to supply a unique data model prefix for non-standard target types. It is the responsibility of the implementer to ensure that there are no namespace collisions when using alternative data models. Refer to the following table for a summary of the OpenC2 TARGET Data Models.

Table 3-6. Target Data Model

Type	Description	Options
data-model	Used to uniquely identify a set of target types so there is no ambiguity; defines the context in which target types are defined.	Choice of: • openc2 • <external-ref>

Targets include objects such as network connections, URLs, hashes, IP addresses, files, processes, and domains. Refer to the following table for a summary of the supported OpenC2 TARGETs in the default TARGET Data Model.

Table 3-7. Summary of Supported Targets

Target Type	Description	Target Specifier
openc2:artifact	The Artifact Object permits capturing an array of bytes (8-bits), as a base64-encoded string or linking to a file-like payload.	mime_type : string, payload_bin : binary, url : string, hashes : hashes-type
openc2:command	The Command Object represents and OpenC2 command.	id : command-ref
openc2:device	The Device Object represents the properties of a hardware device.	description: string, device_type: string, manufacturer: string, model : string, serial_number : string, firmware_version : string
openc2:directory	The Directory Object represents the properties common to a file system directory.	path : string, path_enc : string, created : timestamp, modified : timestamp, accessed : timestamp, contains_refs : list of type object-ref
openc2:disk	The Disk Object represents a disk drive.	disk_name : string, disk_size : integer, free_space : integer, partition_list : list of type disk-partition type : string

openc2:disk-partition	The Disk Partition Object represents a single partition of a disk drive.	created : timestamp, device_name : string, mount_point : string, partition_id : string, partition_length : integer, partition_offset : integer, space_left : integer, space_used : integer, total_space : integer, type : string
openc2:domain-name	The Domain Name represents the properties of a network domain name.	value : string, resolves_to_refs : list of type object-ref
openc2:email-addr	The Email Address Object represents a single email address.	value : string, display_name : string, belongs_to_ref : object-ref
openc2:email-message	The Email Message Object represents an instance of an email message, corresponding to the internet message format described in RFC 5322 and related RFCs.	is_multipart : boolean, date : timestamp, content_type : string, from_ref : object-ref, sender_ref : object-ref, to_refs : list of type object-ref, cc_refs : list of type object-ref, bcc_refs : list of type object-ref, subject : string, received_lines : list of type string, additional_header_fields : dictionary, body : string, body_multipart : list of type mime-part-type, raw_email_ref : object-ref
openc2:file	The File Object represents the properties of a file.	extensions : dictionary, hashes : hashes-type, size : integer, name : string, name_enc : string,

		magic_number_hex : hex, mime_type : string, created : timestamp, modified : timestamp, accessed : timestamp, parent_directory_ref : object-ref, is_encrypted : boolean, encryption_algorithm : open-vocab, decryption_key : string, contains_refs : list of type object-ref, content_ref: object-ref
openc2:ipv4-addr	The IPv4 Address Object represents one or more IPv4 addresses expressed using CIDR notation.	value : string, resolves_to_refs : list of type object-ref, belongs_to_refs : list of type object-ref
openc2:ipv6-addr	The IPv6 Address Object represents one or more IPv6 addresses expressed using CIDR notation.	value : string, resolves_to_refs : list of type object-ref, belongs_to_refs : list of type object-ref
openc2:mac-addr	The MAC Address Object represents a single Media Access Control (MAC) address.	value : string
openc2:memory	The Memory Object represents memory objects.	hashes : list of type string, name : string, memory_source : string, region_size : integer, block_type : string, region_start_address : string, region_end_address : string, extracted_features : string
openc2:network-traffic	The Network Traffic Object	extensions : dictionary, start : timestamp,

	represents arbitrary network traffic that originates from a source and is addressed to a destination.	end : timestamp, is_active : boolean, src_ref : object-ref, dst_ref : object-ref, src_port : integer, dst_port : integer, protocols : list of type string, src_byte_count : integer, dst_byte_count : integer, src_packets : integer, dst_packets : integer, ipfix : dictionary, src_payload_ref : object-ref, dst_payload_ref : object-ref, encapsulates_refs : list of type object-ref, encapsulated_by_ref : object-ref
openc2:openc2	The OpenC2 Object is a subset of the Artifact Object that represents an Actuator's OpenC2 supported capabilities.	value : string, attributes : list of type string, search : string
openc2:process	The Process Object represents common properties of an instance of a computer program as executed on an operating system.	extensions : dictionary, is_hidden : boolean, pid : integer, name : string, created : timestamp, cwd : string, arguments : list of type string, encironment_variables : dictionary, opened_connection_refs : list of type object-ref, creator_user_ref : object-ref, binary_ref : object-ref, parent_ref : object-ref, child_refs : list of type object-ref
openc2:software	The Software	name : string,

	Object represents high-level properties associated with software, including software products.	cpe : string, language : string, vendor : string, version : string
openc2:url	The URL Object represents the properties of a uniform resource locator (URL).	value : string
openc2:user-account	The User Account Object represents an instance of any type of user account, including but not limited to operating system, device, messaging service, and social media platform accounts.	extensions : dictionary, user_id : string, account_login : string, account_type : open-vocab, display_name : string, is_service_account : boolean, is_privileged : boolean, can_escalate_privs : boolean, is_disabled : boolean, account_created : timestamp, account_expires : timestamp, password_last_changed : timestamp, account_first_login : timestamp, account_last_login : timestamp
openc2:user-session	The User Session Object represents a user session.	effective_group : string, effective_group_id : string, effective_user : string, effective_user_id : string, login_time : timestamp, logout_time : timestamp
openc2:volume	The Volume Object represents a generic drive volume.	name : string, device_path : string, file_system_type : string, total_allocation_units : integer, sectors_per_allocation_unit : integer, bytes_per_sector : integer, actual_available_allocation_units : integer,

		creation_time : timestamp, file_system_flag_list : list of type string, serial_number : string
openc2:windows-registry-key	The Registry Key Object represents the properties of a Windows registry key.	key : string, values : list of type windows-registry-value-type, modified : timestamp, creator_user_ref : object-ref, number_of_subkeys : integer
openc2:x509-certificate	The X509 Certificate Object represents the properties of an X.509 certificate, as defined by ITU recommendation X.509.	is_self_signed : boolean, hashes : hashes-type, version : string, serial_number : string, signature_algorithm : string, issuer : string, validity_not_before : timestamp, validity_not_after : timestamp, subject : string, subject_public_key_algorithm : string, subject_public_key_modulus : string, subject_public_key_exponent : integer, x509_v3_extensions : x509-v3-extensions-type

3.5 Actuator Vocabulary

An ACTUATOR is the entity that puts command and control into motion or action. The ACTUATOR executes the ACTION on the TARGET. The ACTUATOR data model is defined in one or more *actuator profiles* where an actuator profile is a document that defines actions that are mandatory to implement, optional and the appropriate actuator specifiers and the actuator specific modifiers. The data model identifies which actuator profile is being referenced. The actuator profiles referenced in this document are for illustrative purposes.

In addition to the default ACTUATOR Data Model, the OpenC2 syntax can support any other data model. To differentiate alternative data models, a data model prefix is used to qualify the actuator type. The default actuator data model will prefix "openc2:" to the actuator

type. The implementer will need to supply a unique data model prefix for non-standard actuator types. It is the responsibility of the implementer to ensure that there are no namespace collisions when using alternative data models. Refer to the following table for a summary of the OpenC2 ACTUATOR Data Models.

Table 3-8. Actuator Data Model

Type	Description	Options
data-model	Used to uniquely identify a set of actuator types so there is no ambiguity; defines the context in which target types are defined.	Choice of: ● openc2 ● <external-ref> ●

Table 3-9. List of Functional Actuators???

Actuator Type	Description
endpoint	Endpoint Device
endpoint-workstation	
endpoint-server	
network	Network Platform
network-firewall	
network-router	
network-proxy	
network-sensor	
network-hips	
network-sense-making	
process	Services/Processes
process-anti-virus-scanner	
process-aaa-service	

process-virtualization-service	
process-sandbox	
process-email-service	
process-directory-service	
process-remediation-service	
process-location-service	

3.6 Modifier Vocabulary

Modifiers provide additional information about the action such as time, periodicity, duration, and location. Modifiers can denote the when, where, and how aspects of an action. The modifier can also be used to convey the need for additional status information about the execution of an action such as a response is required. The requested status/information will be carried in a RESPONSE. Refer to Section 4.6.

Modifiers are similar to specifiers in that they can provide additional context specific details for an action. Modifiers that are applicable to any action are referred to as 'universal modifiers' and are presented in table 3-10. Modifiers that are applicable to a particular action , regardless of the actuator are referred to as , 'Action-specific' and are are identified in the sections detailing out each action. Modifiers that are only applicable to an action for a particular actuator are referred to as 'Actuator Specific' and are defined within the actuator profiles.

The following table lists the set of universal modifiers that are applicable to all types of actions.

Table 3-10. Summary of Universal Modifiers

Modifier	Type	Description	Target Applicability
context	string	A reference that provides context for the action.	All
datetime	date-time (RFC 3339)	The specific date/time to initiate the action.	All

delay	duration (RFC 3339)	The time to wait before performing the action.	All
duration	duration (RFC 3339)	The period of time that an action is valid.	All
id	command_id	The unique identifier for the action.	All
response	ack, status	Indicate the type of response required for the action.	All
respond-to	string	The location where the response should be sent.	All

4. EXAMPLE OpenC2 USAGE

This section provides examples of OpenC2 commands that correspond to each OpenC2 action and its applicable targets. This section also defines any action specific modifiers. The purpose of this section is to provide sample commands that are consistent with the syntax defined in this document and to illustrate the flexibility of the OpenC2 language. Additional examples are presented in in Appendix A.

4.1 Actions that Control Information

These actions are used to gather information needed to further determine courses of action or assess the effectiveness of courses of action. These actions can be used to support data enrichment use cases and maintain situational awareness. These actions typically do not impact the state of the target and are normally not detectable by external observers.

4.1.1 SCAN

4.1.2 LOCATE

4.1.3 QUERY

4.1.4 REPORT

4.1.5 NOTIFY

4.2 Actions that Control Permissions

These actions are used to control permissions and accesses.

4.2.1 DENY

4.2.2 CONTAIN

4.2.3 ALLOW

4.3 Actions that Control Activities/Devices

These actions are used to execute some task, adjust configurations, set and update parameters etc. These actions typically change the state of the system.

4.3.19 COPY

4.3.20 SYNC

4.4 Sensor-Related Actions

These actions are used to control the activities of a sensor in terms of how to collect and provide the sensor data.

4.4.1 DISTILL

4.4.2 AUGMENT

4.5 Effects-Based Actions

Effects-based actions are at a higher level of abstraction and focus on the desired impact rather than a command to execute specific tasks within an enclave. These actions enable the coordination actions, while permitting a local enclave to execute actions in accordance with its local policies and/or capabilities. .

Implementation of an effects-based action requires that the recipient enclave has a decision making capability because an effects-based action permits multiple possible responses.

4.5.1 INVESTIGATE

4.5.2 MITIGATE

4.5.3 REMEDIATE

4.6 Response and Alert

RESPONSE is used to provide data requested as a result of an action. The RESPONSE message will contain the requested data and have a reference to the action that initiated

the response. ALERT is used to signal the occurrence of an event or error. It is an unsolicited message that does not reference a previously issued action.

4.6.1 RESPONSE

4.6.2 ALERT

5. Example OpenC2 Use Case

Appendix A. Example OpenC2 Commands

[Document Note: These subsections in this appendix link to Appendix Sections in the Action Detail sub-documents. It is the author's intention that the Appendix Sections be located in this appendix in the final document.]

A.1 ALERT

A.2 ALLOW

A.3 AUGMENT

A.4 CANCEL

A.5 CONTAIN

A.6 COPY

A.7 DELAY

A.8 DELETE

A.9 DENY

A.10 DETONATE

A.11 DISTILL

A.12 INVESTIGATE

A.13 LOCATE

A.14 MITIGATE

A.15 MOVE

A.16 NOTIFY

A.17 PAUSE

A.18 QUERY

A.19 REDIRECT

A.20 REMEDIATE

A.21 REPORT

A.22 RESPONSE

A.23 RESTART

A.24 RESTORE

A.25 RESUME

A.26 SAVE

A.27 SCAN

A.28 SET

A.29 SNAPSHOT

A.30 START

A.31 STOP

A.32 SUBSTITUTE

A.33 SYNC

A.34 THROTTLE

A.35 UPDATE