SCAPv2 and OpenC2

David Kemp, NSA Cybersecurity

SCAPv2 Fall Virtual Workshop

29 Sept 2020

Relationship

- SCAP defines assessment data
- SCAPv2 includes ongoing assessment operations
- OpenC2 protocol communicates assessment requests and results
- Development Projects:
    - SCAPv2 Endpoint Data Collection Prototype
    - OpenC2 SBOM Proof of Concept

NIST

Information Technology Laboratory

COMPUTER SECURITY RESOURCE CENTER

PROJECTS | SECURITY CONTENT AUTOMATION PROTOCOL

Security Content Automation Protocol SCAP

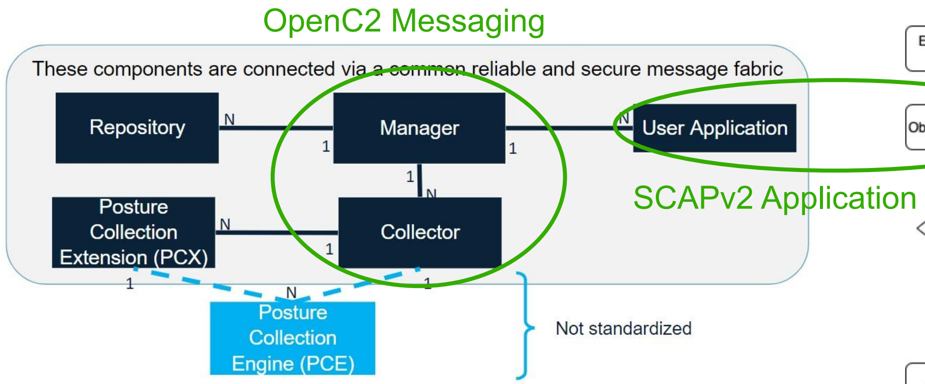
OASIS

Open Command and Control (OpenC2) Language Specification Version 1.0

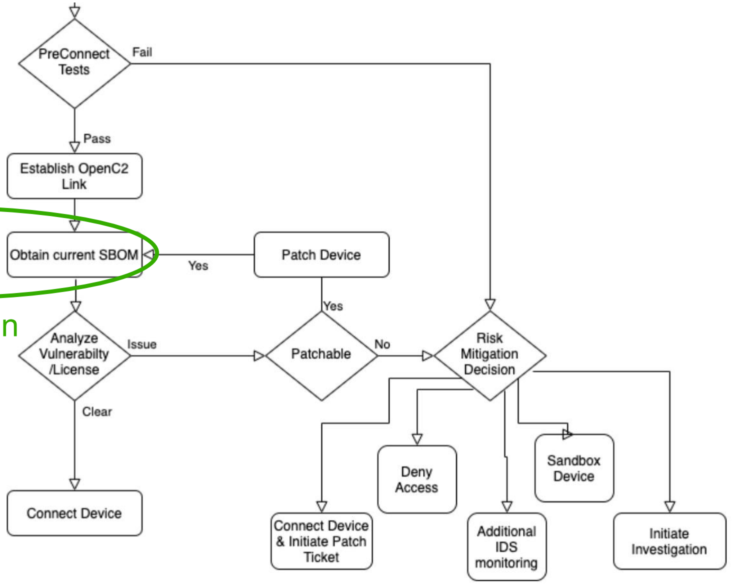
Committee Specification 02

24 November 2019

Development Projects


OpenC2 Messaging
These components are connected via a common reliable and secure message fabric
Repository
Manager
User Application
Posture Collection Extension (PCX)
Collector
Posture Collection Engine (PCE)
N
1
Not standardized
SCAPv2 Application

SCAPv2 Data Collection Prototype



OpenC2 Software Bill of Materials Proof of Concept

https://github.com/oasis-tcs/openc2-usecases/tree/master/SBOM-PoC

OpenC2 Scope

Remediation Cycle
(OODA Loop):

- Sense
- Analyze
- Decide
- Act

controls

OpenC2: vendor-agnostic vocabulary of core and extension commands, integrable into response playbooks.

- NETCONF vendor extensions
- OpenC2 **function** extensions

The Struggle Behind Long Remediation Cycles

A 2018 incident response survey by SANS showed an increase in the number of organizations detecting incidents within 24 hours, along with a general move to shorter detection. However, despite 53% of organizations detecting incidents within 24 hours, 61% took two or more days to remediate.

Without context, isolated events don't have much meaning and only add to alert fatigue, while data enrichment from correlating multiple sources reduces the scope of your investigation so you can focus faster on the real threat. Cisco Threat Response brings together data from across the architecture into one console. It gives you a data-enrichment tool for a more comprehensive story across multiple vectors.

Out of scope

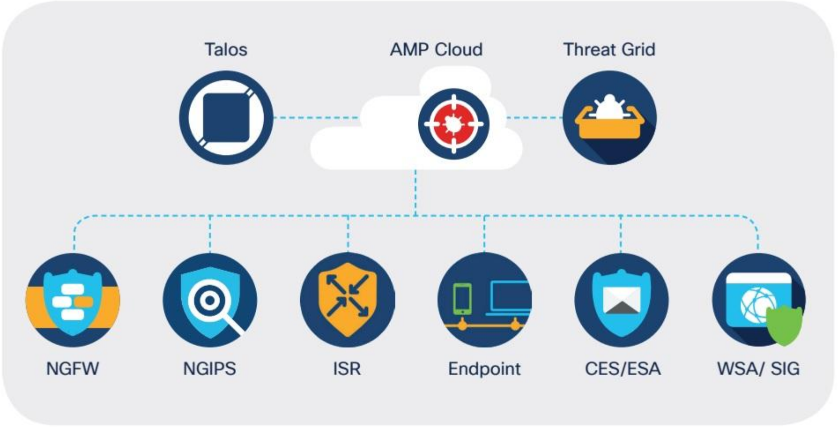
Security that
Works Together

In scope

See once, block everywhere

Investigate and respond to threats across network, web, email and endpoints

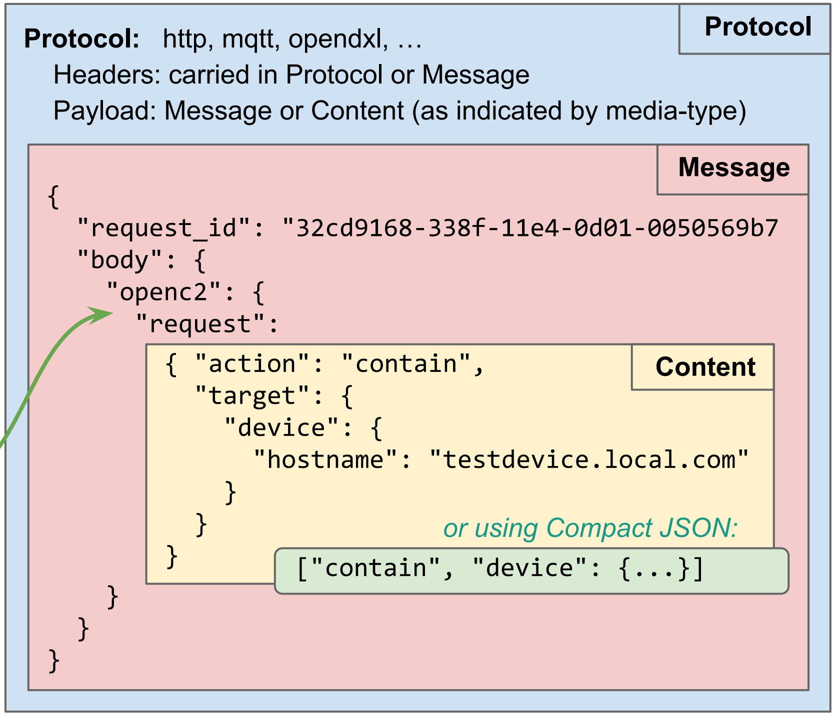
Drive zero-trust for organizations with SecOps journey well underway



https://www.cisco.com/c/dam/en/us/products/collateral/security/amp-for-endpoints/amp4e-w-orbital-wp.pdf

What Is OpenC2?

- **Content:** control structure and cybersecurity vocabulary for discrete **actions**
  - **Action** (verb): allow, deny, contain, scan, query, restart, …
  - **Target** (noun): device, file, ip address, ip connection, url, …
  - **Args**: additional details: where and how to perform the action
  - **Actuator** (function): packet filtering, intrusion prevention, assessment, ...
- **Message:** content-agnostic payload structure
  - Headers
  - Content-type (openc2), Message-type (request, response, notification)
  - Content
- **Protocol:** message bindings for transport protocols
  - HTTPS, MQTT, OpenDXL, ...
- **Information Modeling Language**
  - abstract syntax for content in JSON, CBOR, XML, ... formats
  - machine-readable schema ⇒ property tables, IDL, UML diagrams, node-edge graphs



Protocol: http, mqtt, opendxl, …
Headers: carried in Protocol or Message
Payload: Message or Content (as indicated by media-type)

```
{
  "request_id": "32cd9168-338f-11e4-0d01-0050569b7
  "body": {
    "openc2": {
      "request":
        { "action": "contain",
          "target": {
            "device": {
              "hostname": "testdevice.local.com"
            }
          }
        }
    }
  }
}
```

or using Compact JSON:

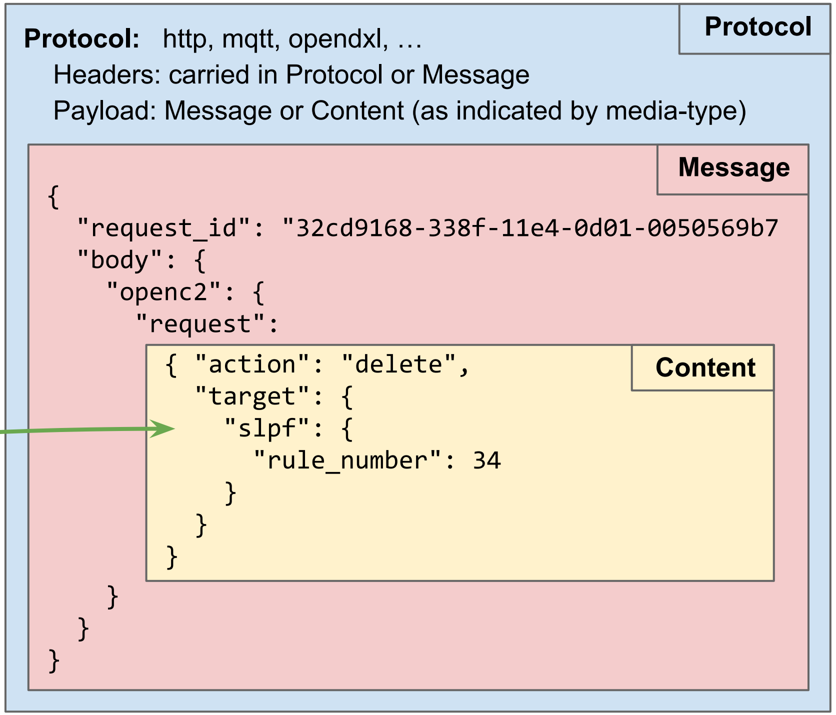
```
["contain", "device": {...}]
```

Friendly Encoding: https://www.w3.org/2011/10/integration-workshop/s/ExperienceswithJSONandXMLTransformations.v08.pdf

OpenC2 Actuator Profiles

- **Language Specification:** defines the Content structure and a cybersecurity vocabulary of simple common objects:
  - **Target** (noun): device, file, ip address, ip connection, url, …
- **Actuator Profiles:** define application-specific objects that may be simple or complex:
  - **Target** (noun):
    - slpf/rule_number
    - scap/assessment (to be defined)
- **Device** (OpenC2 Consumer) supports parts of the core **Language** and one or more **Actuator Profiles**

https://github.com/oasis-tcs/openc2-usecases/tree/master/SBOM-PoC/Schemas



Protocol: http, mqtt, opendxl, …
Headers: carried in Protocol or Message
Payload: Message or Content (as indicated by media-type)

```
{
  "request_id": "32cd9168-338f-11e4-0d01-0050569b7
  "body": {
    "openc2": {
      "request":
        { "action": "delete",
          "target": {
            "slpf": {
              "rule_number": 34
            }
          }
        }
    }
  }
}
```

or: "How to OpenC2-ize a system spec"

Conceptual Design (messages)

SCAPv2

Data Collection Experiment

List each Message sent between components

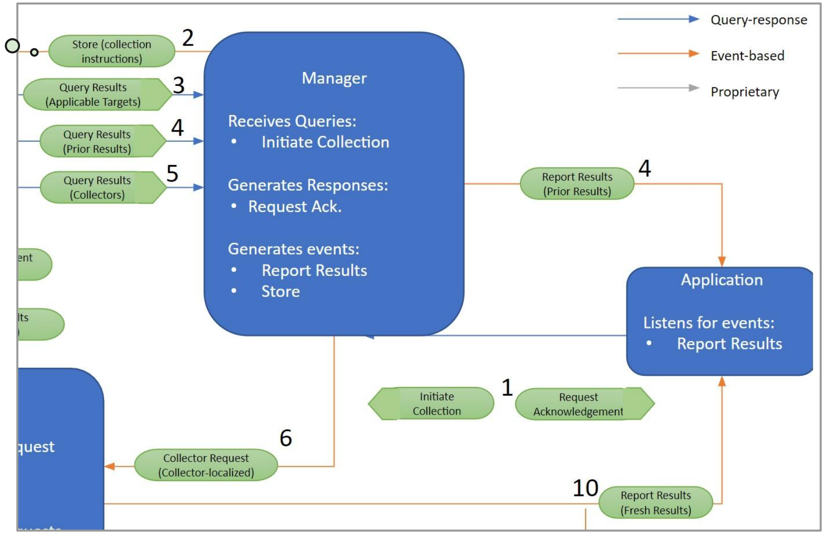
1a. Initiate Collection
1b. Request Acknowledgement

2. Store (collection instructions)

3a. Query (Applicability)
3b. Query Results (Applicable Targets)

Every Message needs a unique name

- Message names used by SCAPv2 team ("Initiate Collection") reflect purpose (verb)
- Content names ("Assessment-Instructions") refer to data structure (noun)



SCAPv2

Data Collection Experiment

Assign Content name used in each Message

1a. **Initiate Collection** is a request message with specified content
1b. **Request Acknowledgement** is a response message, no content

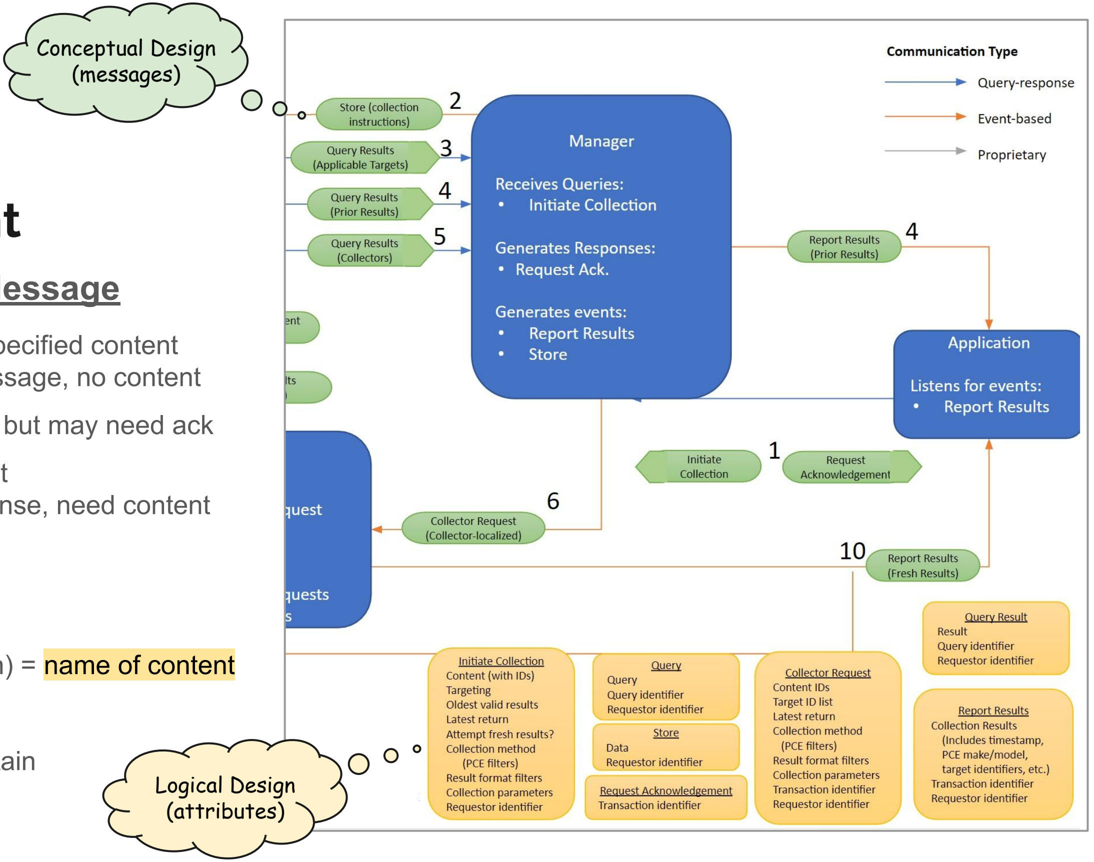
2. **Store (collection instructions)** shown as event, but may need ack

3a. **Query (Applicability)** is a request, need content
3b. **Query Results (Applicable Targets)** is a response, need content

4a. …

For each Message:
* pick an Action(verb) for each request. Target(noun) = name of content
* define content types and attributes

Example: 1a: Initiate Collection message might contain
Action = scan, **Target** = Assessment-Instructions



Conceptual Design (messages)
Logical Design (attributes)
Communication Type
Query-response
Event-based
Proprietary
Manager
Receives Queries:
• Initiate Collection
Generates Responses:
• Request Ack.
Generates events:
• Report Results
• Store
Application
Listens for events:
• Report Results
2 Store (collection instructions)
3 Query Results (Applicable Targets)
4 Query Results (Prior Results)
5 Query Results (Collectors)
4 Report Results (Prior Results)
1 Initiate Collection
Request Acknowledgement
6 Collector Request (Collector-localized)
10 Report Results (Fresh Results)
ent
lts
quest
quests
s
Initiate Collection: Content (with IDs), Targeting, Oldest valid results, Latest return, Attempt fresh results?, Collection method (PCE filters), Result format filters, Collection parameters, Requestor identifier
Query: Query, Query identifier, Requestor identifier
Store: Data, Requestor identifier
Request Acknowledgement: Transaction identifier
Collector Request: Content IDs, Target ID list, Latest return, Collection method (PCE filters), Result format filters, Collection parameters, Transaction identifier, Requestor identifier
Query Result: Result, Query identifier, Requestor identifier
Report Results: Collection Results (Includes timestamp, PCE make/model, target identifiers, etc.), Transaction identifier, Requestor identifier

SCAPv2 Actuator Profile

Physical Design
(information model
with types)

Initiate Collection message:
  Target Name: Assessment-Instructions
  Target Type: Record (JSON object)

Initiate Collection
Content (with IDs)
Targeting
Oldest valid results
Latest return
Attempt fresh results?
Collection method
  (PCE filters)
Result format filters
Collection parameters
Requestor identifier

```
Assessment-Instructions = Record
   1 content      Assessment-Content
   2 targets      Assessment-Target [1..*]
   3 oldest       DateTime
   4 latest       DateTime
   5 refresh      Boolean
   6 methods      PCE-Filter [1..*]
   7 formats      Result-Format [1..*]
   8 params       Collection-Parameters
   9 requestor    Requestor-ID

Assessment-Content = String
Assessment-Target = String
PCE-Filter = String
Result-Format = String
Collection-Parameters = String
Requestor-ID = Binary /uuid
```

OpenC2 JADN Interface Definition Language

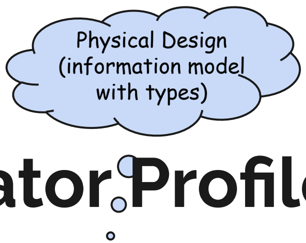
Make up attribute names and types

"Content (with IDs)" is from the system description
"content" is a (made-up) attribute name (could be better)
"Assessment-Content" is a (made-up) type name

"Targeting" sounds vaguely plural, so give it multiplicity [1..*]
"Oldest valid results" sounds like a date-time, but may be more
Create stubs (String) for unknown types
Guess that "Requestor identifier" might be a UUID

Do the same for all Messages

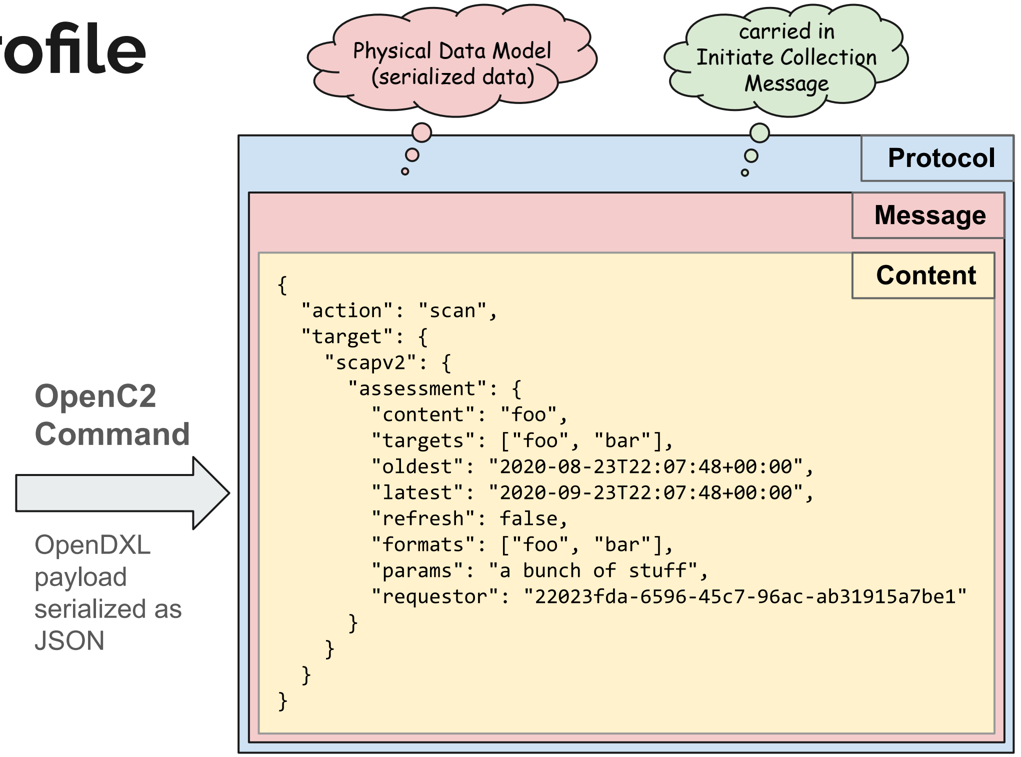
OpenC2 schema for Data Collection Experiment is now defined in sufficient detail for experimenting. Fill in stubs later.



SCAPv2 Actuator Profile

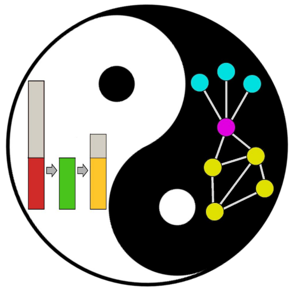
```
Assessment-Instructions = Record
   1 content      Assessment-Content
   2 targets      Assessment-Target [1..*]
   3 oldest       DateTime
   4 latest       DateTime
   5 refresh      Boolean
   6 methods      PCE-Filter [1..*]
   7 formats      Result-Format [1..*]
   8 params       Collection-Parameters
   9 requestor    Requestor-ID

Assessment-Content = String
Assessment-Target = String
PCE-Filter = String
Result-Format = String
Collection-Parameters = String
Requestor-ID = Binary /uuid
```

OpenC2 JADN Interface Definition Language


OpenC2 Command
OpenDXL payload serialized as JSON
Physical Data Model (serialized data)
carried in Initiate Collection Message
Protocol
Message
Content

```
{
  "action": "scan",
  "target": {
    "scapv2": {
      "assessment": {
        "content": "foo",
        "targets": ["foo", "bar"],
        "oldest": "2020-08-23T22:07:48+00:00",
        "latest": "2020-09-23T22:07:48+00:00",
        "refresh": false,
        "formats": ["foo", "bar"],
        "params": "a bunch of stuff",
        "requestor": "22023fda-6596-45c7-96ac-ab31915a7be1"
      }
    }
  }
}
```

OpenC2 Information Modeling



- OpenC2 created the JADN formal information modeling language based on **information theory** and **graph theory**.
  - A **package** is a **namespace** for a collection of **type definitions**
  - Every type definition is a graph node
  - Types define the **information** (entropy) contained in data instances
  - **Serialization rules** define data formats used to represent information instances
- Graph structure:
  - Encourages normalization and reuse of named types
  - Supports evolution from conceptual design to concrete schemas.
- An IM is used to both generate specifications and validate data
  - Improves quality and ensures consistency
  - IM is a **normative** definition, not just "a representation of concepts"

https://github.com/oasis-tcs/openc2-jadn/blob/working/jadn-v1.0-wd01.md

An **information model** is a representation of concepts and the relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse.

Information theory studies the quantification, storage, and communication of information.

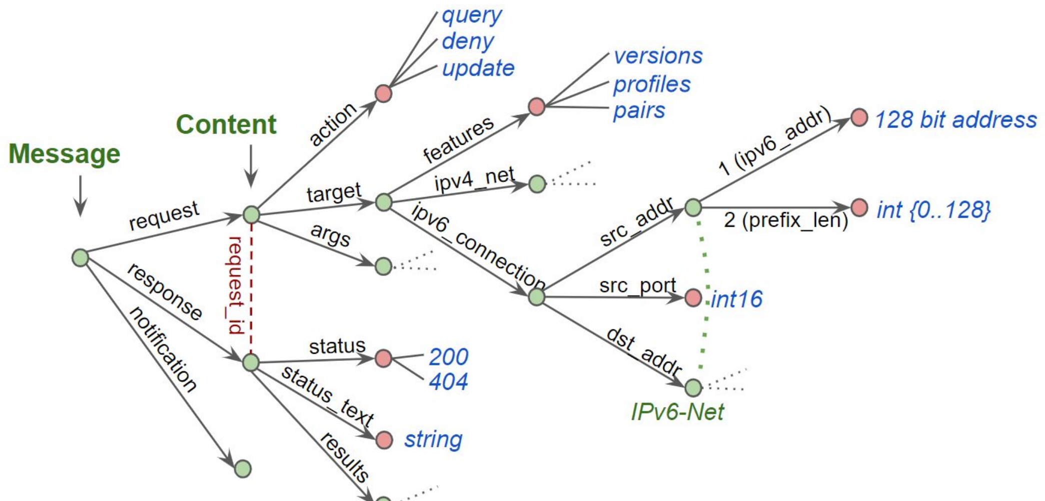
Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects.

--- Wikipedia

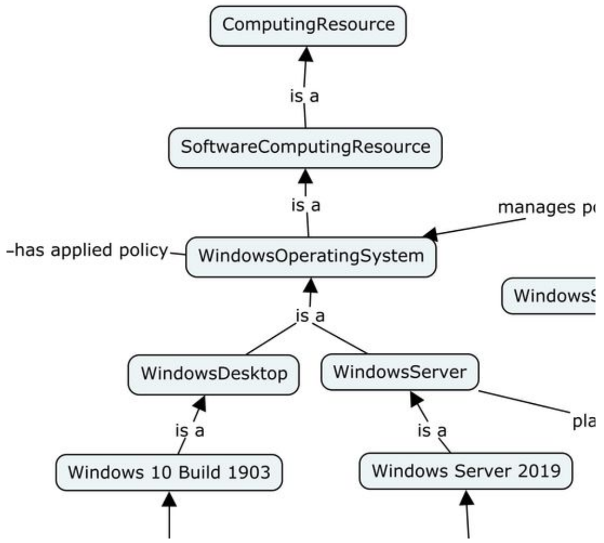
OpenC2 Graph API

- Openc2 defines an attribute graph
  - graph can be denormalized to a directed tree
- A graph bound to a protocol is an API
  - one graph can be bound to multiple protocols
- Falcor: ***"The data is the API"***:
  - protocol payload is the entire graph
- REST API: graph is split between resource URLs and payload sub-graphs
  - resource supports only CRUD methods: (POST, GET, PUT, PATCH, DELETE)
  - graph API supports any noun or verb (request, notification, scan, stop, start, …)

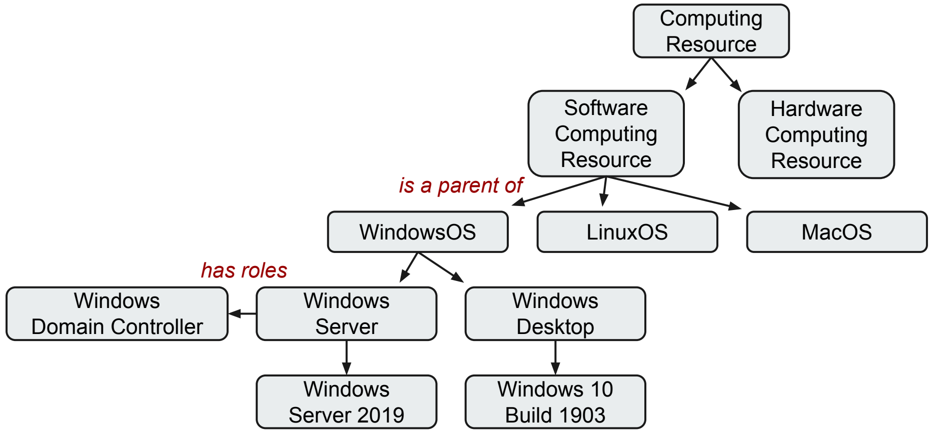
Falcor: https://netflix.github.io/falcor/
GraphQL: https://graphql.org/



Relationship Modeling



https://lists.oasis-open-projects.org/g/oca-architecture-wg/message/42



OpenC2 models **data relationships** as **Directed Acyclic Graphs**

- convert "is a" to "is a parent of" -- identifies alternatives
- undirected links support arbitrary relationships among data instances

```
ComputingResource = Choice
  1 sw   SoftwareComputingResource
  2 hw   HardwareComputingResource

WindowsServer = Record
  1 version  WindowsServerVersion
  2 roles    WindowsServerRole [1..*]
```

OpenC2 JADN IDL

UML Generalization Sets:

- *disjoint / overlapping*
- *complete / incomplete*

"parent-of" = Choice = disjoint

Goal is to be complete