SCHEMA OVERVIEW

May 2017

Dave Kemp -- NSA, Cybersecurity Architecture and Strategies

WannaCry?

OpenC2?


```
{
  "action": "scan",
  "target": {
    "device": {
        "model": "bar“
    },
    "author": "Charles Dickens",
    "title": "A Tale of Two Cities",
    "quote": "We had everything before us, we had nothing before us, "
        "we were all going direct to Heaven, "
        "we were all going direct the other way – "
        "in short, the period was so far like the present period, "
        "that some of its noisiest authorities insisted on "
        "its being received, for good or for evil, in the "
        "superlative degree of comparison only."
  }
}
```

STIX

Messages Defined using Property Tables

Relationships from the Indicator can describe the malicious or suspicious behavior that it directly detects (Malware, Tool, and Attack Pattern) as well as the Campaigns, Intrusion Sets, and Threat Actors that it might indicate the presence of.

2.5.1 Properties

Common Properties
`type`, `id`, `created_by_ref`, `created`, `modified`, `revoked`, `labels`, `external_references`, `object_marking_refs`, `granular_markings`
Indicator Specific Properties
`name`, `description`, `pattern`, `valid_from`, `valid_until`, `kill_chain_phases`

Property Name	Type	Description
`type` (required)	`string`	The value of this property **MUST** be `indicator`.
`labels` (required)	`list` of type `open-vocab`	This property is an Open Vocabulary that specifies the type of indicator. This is an open vocabulary and values **SHOULD** come from the `indicator-label-ov` vocabulary.
`name` (optional)	`string`	A name used to identify the Indicator.
`description` (optional)	`string`	A description that provides more details and context about the Indicator, potentially including its purpose and its

OpenC2

JAEN

```
["OpenC2Command", "Record", [], "", [
   [1, "action", "Action", [], ""],
   [2, "target", "Target", [], ""],
   [3, "actuator", "Actuator", ["?"], ""],
   [4, "modifiers", "Modifiers", ["?"], ""]]],

 ["OpenC2Response", "Record", [], "", [
   [1, "status", "status-code", [], "Adapted from HTTP Status Codes, RFC 7231"],
   [2, "statusText", "String", ["?"], "Status description"],
   [3, "response_src", "device-id", ["?"], "ID of the responder/actuator"],
   [4, "command_id", "command-id", ["?"], "Command unique identifier, from \"command_id\" modifier"],
   [5, "results", "String", ["?"], "Results of executing the command"]]],
```

Messages defined using a JSON document (schema) from which property tables can be generated.

	A	B	C	D	E	F
10						
11	**Type Name:**	OpenC2Command				
12	**Type:**	Record				
13						
14		**Id**	**Name**	**Type**	**Description**	
15		1	action (required)	Action (vocab)		
16		2	target (required)	Target (Choice)		
17		3	actuator (optional)	Actuator (Choice)		
18		4	modifiers (optional)	Modifiers (Map)		
19						
20						
21	**Type Name:**	OpenC2Response				
22	**Type:**	Record				
23						
24		**Id**	**Name**	**Type**	**Description**	
25		1	status (required)	status-code (vocab)	Adapted from HTTP Status Codes, RFC 7231	
26		2	statusText (optional)	String	Status description	
27		3	response_src (optional)	device-id (String)	ID of the responder/actuator	
28		4	command_id (optional)	command-id (String)	Command unique identifier, from "command_id" modifier	
29		5	results (optional)	String	Results of executing the command	
30						

Meta | Types | Vocab

Purpose of a Schema Language

- Standards Authors
  - Unambiguous language specification
  - Automated checking of example messages
- Standards Consumers
  - Schema-based message validation without writing code
  - Interoperability testing with hand-coded apps
  - Automatic bandwidth optimization options

Machine-Readable Property Tables (JAEN)

- JSON document that defines an abstract schema
  - Import directly by applications,
  - Translate to concrete schemas used by applications, or
  - Translate to property tables used by application programmers

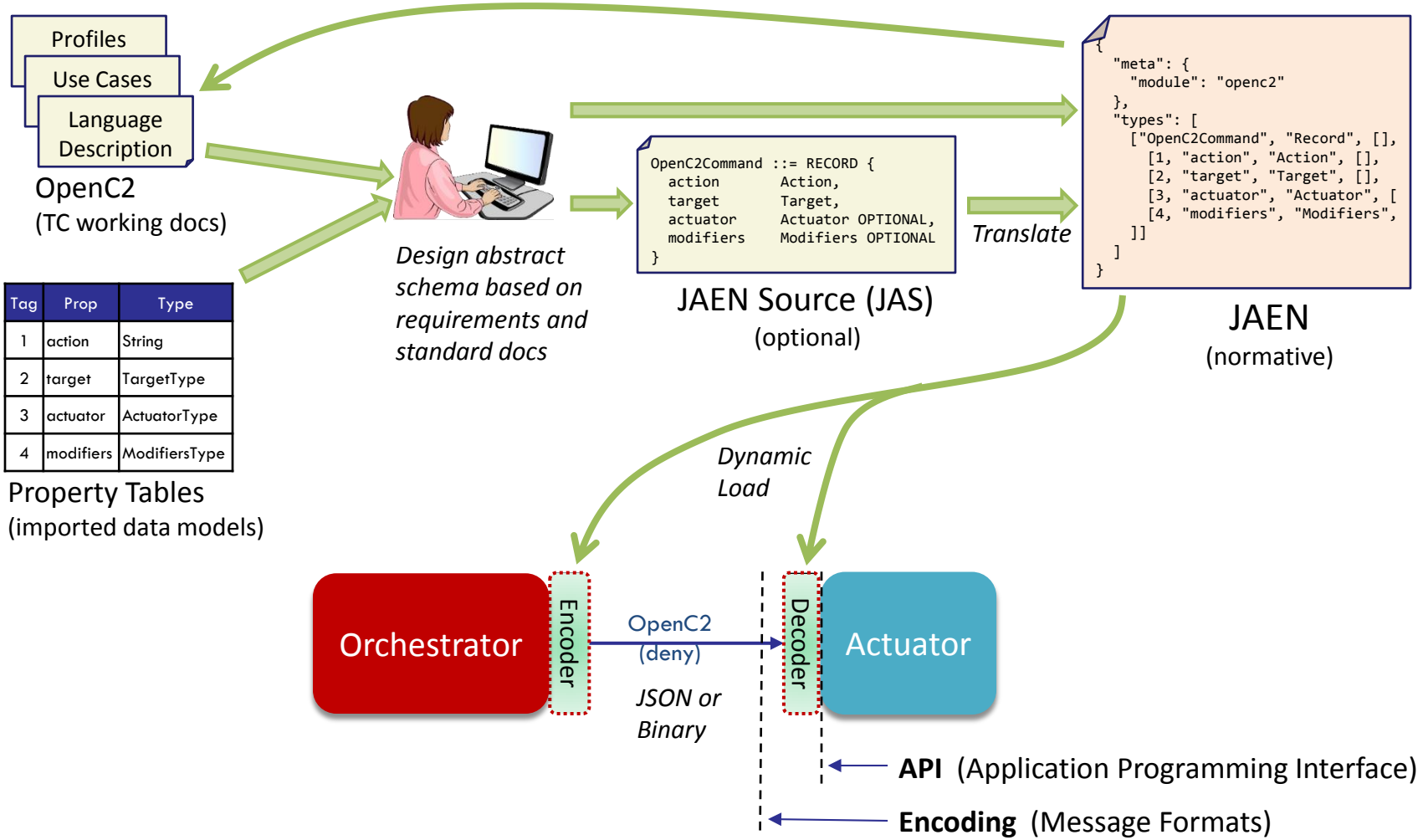
Two sections:
- config information
- datatype definitions

Definitions:
- datatype, base type, opts, desc

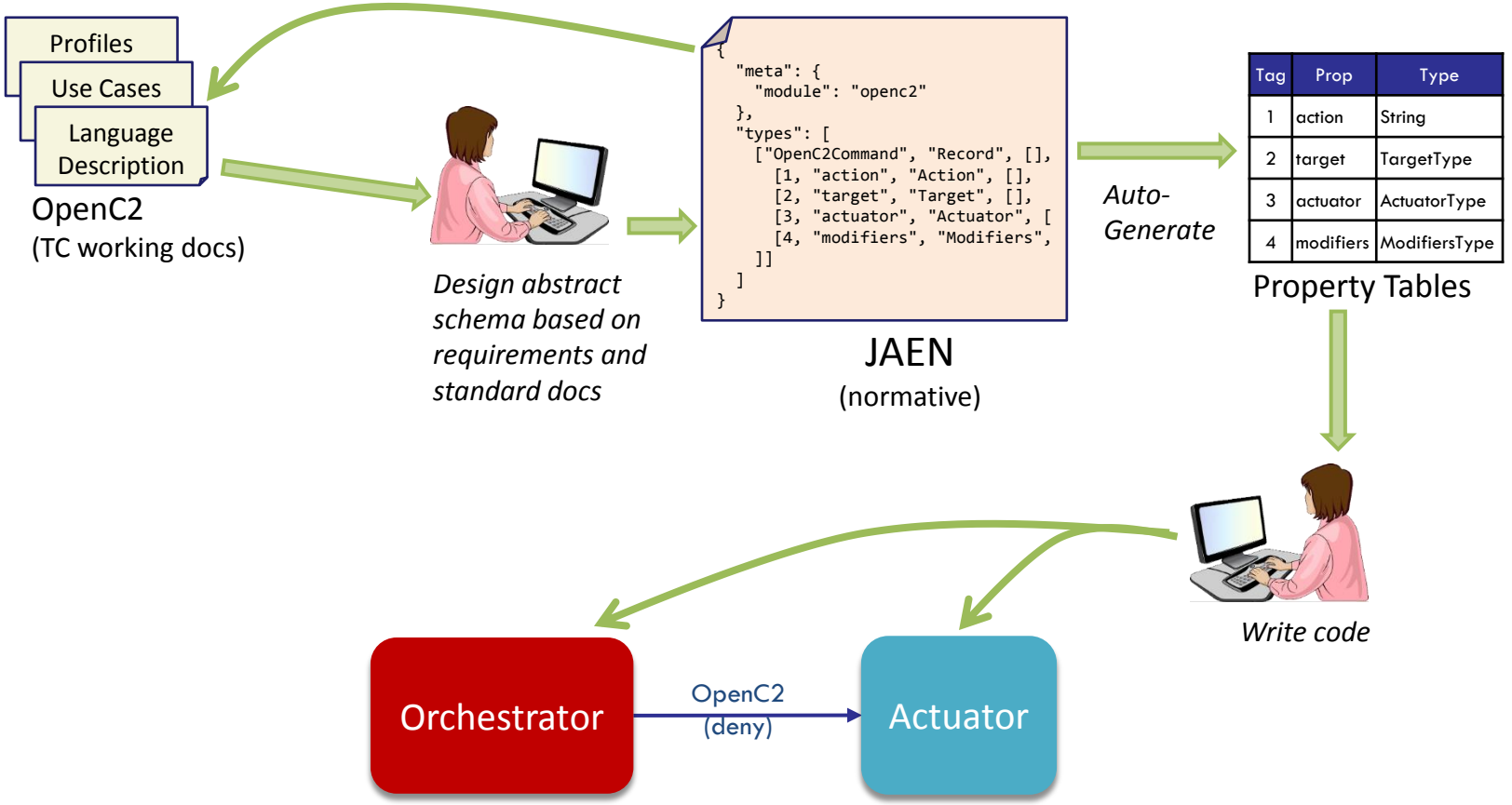
Fields:
- position (ordinal) or tag
- field name
- datatype
- options
- description

```
{
  "meta": {
    "module": "openc2"
  },
  "types": [
    ["OpenC2Command", "Record", [], "", [
      [1, "action", "Action", [], ""],
      [2, "target", "Target", [], ""],
      [3, "actuator", "Actuator", ["?"], ""],
      [4, "modifiers", "Modifiers", ["?"], ""]
    ]]
  ]
}
```

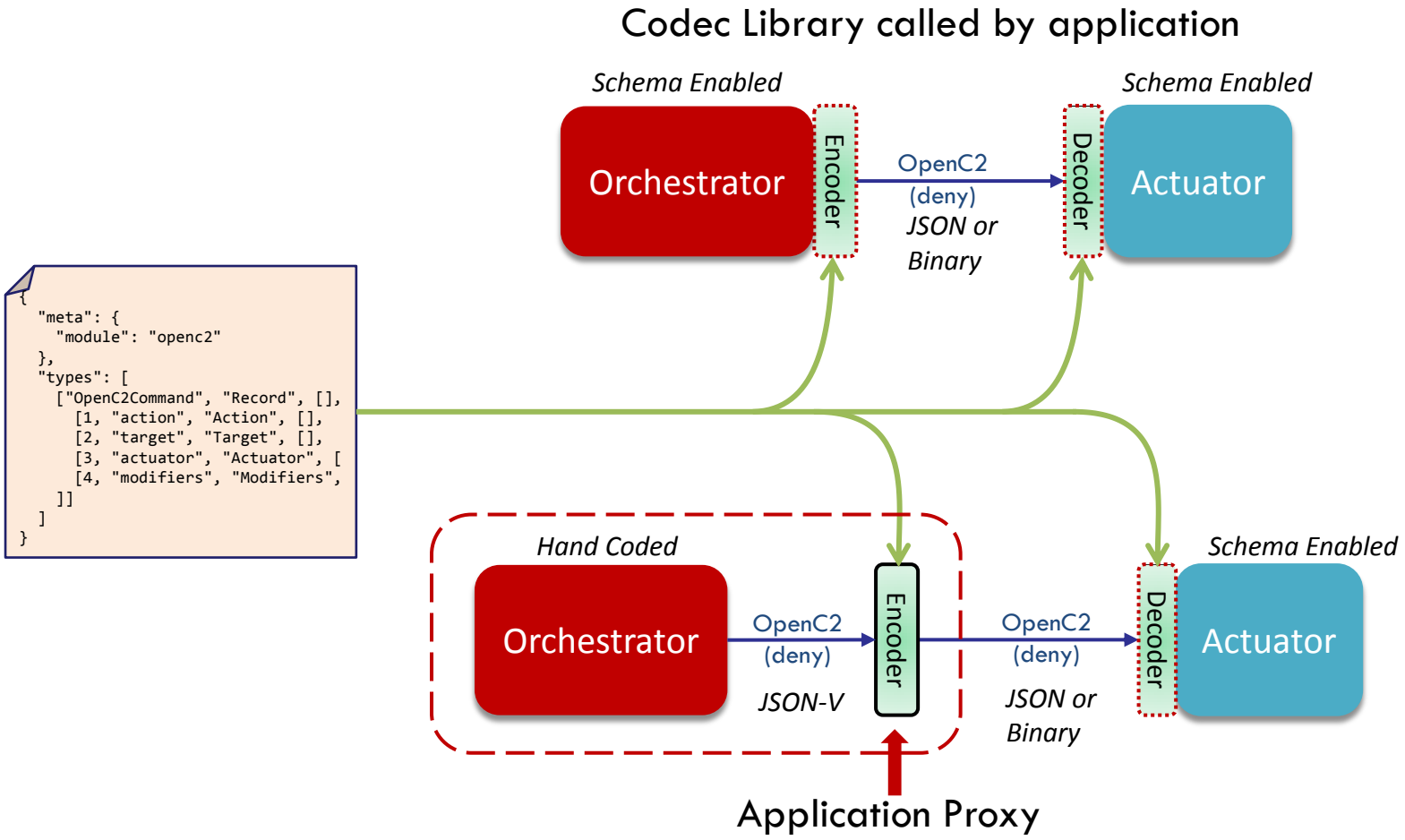
Schema Design and Use



Schema Design and Use (hand-coding)



One API


Codec Library called by application
Schema Enabled
Schema Enabled
Orchestrator
Encoder
OpenC2
(deny)
JSON or
Binary
Decoder
Actuator
{
"meta": {
"module": "openc2"
},
"types": [
["OpenC2Command", "Record", [],
[1, "action", "Action", [],
[2, "target", "Target", [],
[3, "actuator", "Actuator", [
[4, "modifiers", "Modifiers",
]]
]
}
Hand Coded
Schema Enabled
Orchestrator
OpenC2
(deny)
JSON-V
Encoder
OpenC2
(deny)
JSON or
Binary
Decoder
Actuator
Application Proxy

One Standard

- JAEN and Property Tables are both serialization-agnostic
  - Permits one product to support multiple serializations
  - Decouples standard specification from MTI decisions

```
{
  "meta": {
    "module": "openc2"
  },
  "types": [
    ["OpenC2Command", "Record", [],
      [1, "action", "Action", [],
      [2, "target", "Target", [],
      [3, "actuator", "Actuator", [
      [4, "modifiers", "Modifiers",
    ]]
  ]
}
```

	A	B	C	D
10				
11	**Type Name:**	OpenC2Command		
12	**Type:**	Record		
13				
14		**Id**	**Name**	**Type**
15		1	action (required)	Action (vocab)
16		2	target (required)	Target (Choice)
17		3	actuator (optional)	Actuator (Choice)
18		4	modifiers (optional)	Modifiers (Map)
19				

Message Structure

- API format uses nested objects
- Can be converted to API-Flat (single object) format
  - Identical information, lossless bidirectional conversion
  - Flat format may be easier for applications to work with

```
{ "action": "scan",
  "target": {
    "domain_name": {
      "value": "www.example.com",
      "resolves_to": [
        {"ipv4": {"value": "198.51.100.2"}},
        {"name": {"value": "ms34.example.com"}}
      ]
    }
  }
}
```

```
{ "action": "scan",
  "target.domain_name.value": "www.example.com",
  "target.domain_name.resolves_to.0.ipv4.value": "198.51.100.2",
  "target.domain_name.resolves_to.1.name.value": "ms34.example.com"
}
```

Summary

- STIX: Standard contains property tables and JSON examples
  - Property tables are *normative* but *not testable*
  - Member-developed JSON Schemas are testable but are not part of the STIX standard
  - No path to efficient message encoding for production systems

- OpenC2: Standard contains schema and examples
  - Abstract schema is *normative* and *testable*
  - Property tables are auto-generated from normative schema
  - Member-developed concrete schemas (JSON and Binary) are derived from abstract schema and are testable
    - Goal: auto-generation of concrete schemas from abstract schema
  - Examples have been validated against normative schema