OpenC2

FACE TO FACE WORKSHOP

29 Sept 2016

9/29/2016 F2F Workshop Agenda

(Page 1 of 2)

Time	Topic	Presenter/Facilitator
08:30 – 09:00	**Introduction and Key Note**	Neal Ziring, MPO
	Prototype Implementations	
09:00 - 9:05	□ Overview	Joyce Fai, General Dynamics
9:05 - 09:35	□ OpenC2 Actions as Multi-methods	Joshua Brule, University of MD
09:35 – 09:50	□ Deny at Perimeter	Larry Salazar, General Dynamics
09:50 – 10:05	□ OpenC2 as a Supported Data Model in Cisco Threat INTEL API	Jyoti Verma, Cisco
10:05 – 10:30	□ NSA/ APL Host based Implementation	Kevin Miller, MPO
10:30 – 10:45	Break	
10:45 – 11:30	□ OpenC2 and Distributed Network Security Policy Convergence	Eric Voit, Cisco
11:30 – 12:00	□ Schema Design	Dave Kemp, MPO
12:00 – 12:30	**Working Lunch**	

9/29/2016 F2F Workshop Agenda

(Page 2 of 2)

Time	Topic	Presenter/Facilitator
	Implementation Considerations	
12:30 – 13:30	❑ Lessons and Implementation Considerations	Group Discussion; Facilitated by Joe Brule (MPO) and Jason Romano (GD)
13:30 – 1400	❑ Implementing Structured COA for STIX Version 2.X	Jyoti Verma, Cisco Bret Jordan, Bluecoat
	Way Forward	
14:00 – 15:00	❑ What is the next REALLY Hard Problem	Group Discussion
15:00 – 15:15	Break	
15:15 – 15:25	❑ OASIS Overview	Bret Jordan, Bluecoat
15:25 – 16:00	❑ Path to Standardization	Group Discussion
16:00 – 16:45	❑ Way Forward for OpenC2	Group Discussion
16:45 – 17:00	**Wrap Up**	Joe Brule, MPO
17:00	**Adjourn**	

OpenC2

QUARTERLY FACE TO FACE WORKSHOP
FALL 2016

Neal Ziring
Technical Director,
NSA Capabilities Directorate

29 SEP 2016

Agenda

- Background
  - Motivation
  - Status
- Way Forward
  - Implementation Considerations
  - Reference Implementations
  - Actuator Profiles
  - Path to Standardization
- Future of the OpenC2 Forum

The Motivation and Vision

- Future Cyber Defense Tactics:
  - Sharing of indicators
  - Coordination of response actions
  - Automated, multi-part actions at machine speed
- OpenC2 Forum
  - Identify and fill gaps as they pertain to command and control for the provision or support of cyberdefense
  - Create a diverse and collaborative environment.
- Standardization is a Key Enabler for Unambiguous C2

OpenC2 ‘Philosophy’

- Pre-existing standards will be leveraged to the greatest extent practical
- Minimize Complexity of Command
  - Minimize Overhead on Sensor/Actuator
  - Facilitate Adoption
- Infrastructure, architecture, and vendor agnostic
- Extensible to support different levels of detail and future technologies

OpenC2 Design Principles

- Lightweight Efficient Machine-to-Machine communications
- Abstract
  - Focuses on ‘What’ to do vice ‘Device Specific’
  - Permits different levels of commanding
- Extensible
  - Enables additional precision and flexibility
- Agnostic
  - e.g., Transport, Authentication, Integrity controls
  - Enables flexibility w.r.t. implementation

Enable Unambiguous Machine-to-Machine Command and Control Messages

Status/ Recently Posted

- Version 1.0 of the Language Description Document
  - Posted on OpenC2.org
  - Define Actions, Syntax and Modular Data Model
- Version 1.0 of the IA Considerations Document
- STIX sub-working group
  - OpenC2 to be included in STIX 2.1
  - Collaboration with STIX WG
- Draft SDN Profile posted
- OpenC2 Schema in process
  - Draft JSON encoded version posted on OpenC2.org
  - ASN.1 version in progress

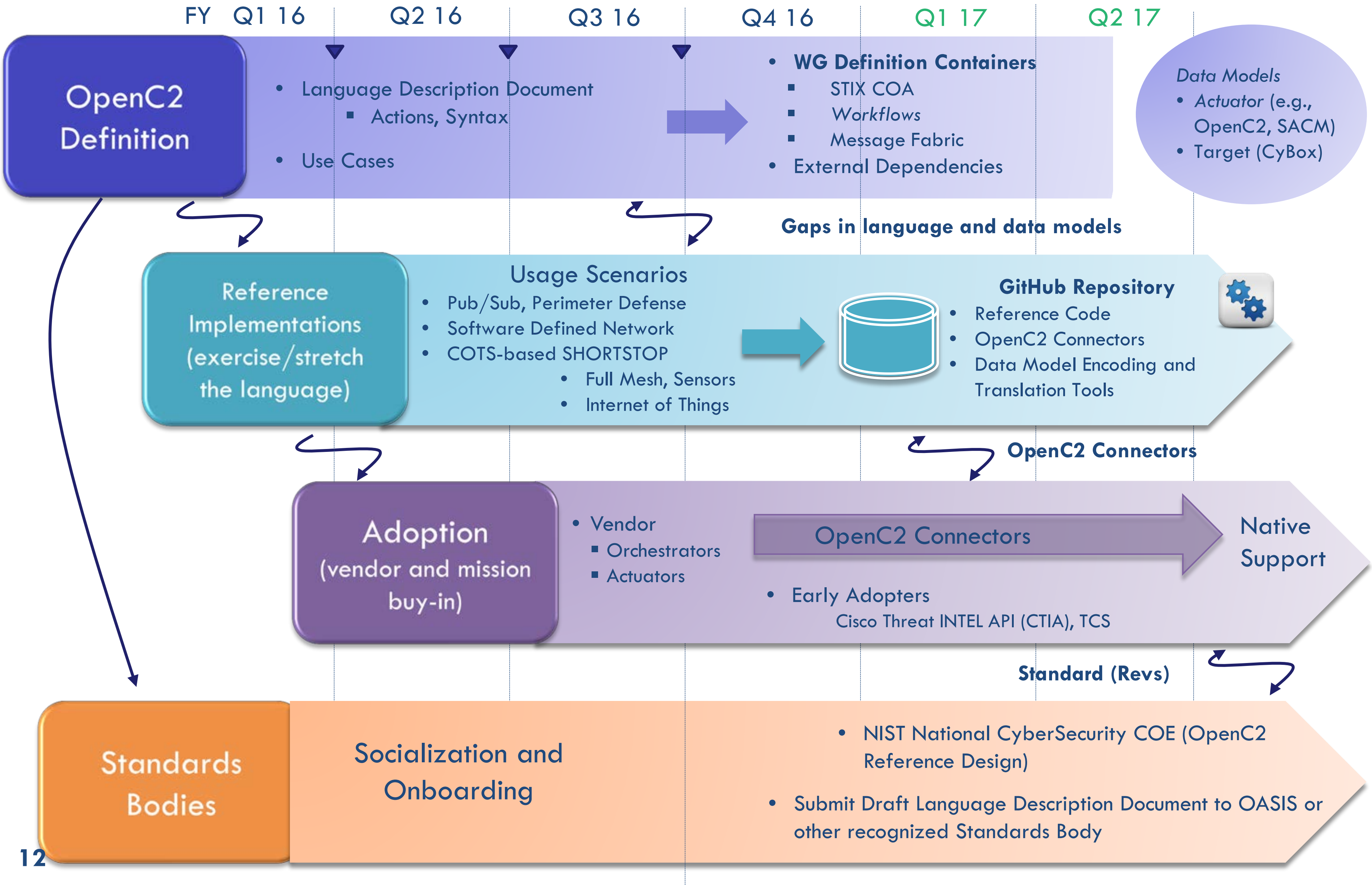
Works In Progress

- Prototype Implementations
  - Multiple Efforts from government, industry, and academia
  - Capturing Lessons Learned
    - Modify Language Description Document
    - Identify/ Prioritize Next Steps
- Expand Documentation
  - Implementation Considerations
  - Annexes to Language Description Document
- More Conspicuous Public Presence
  - Presented at multiple forums
  - Expanded Public facing website
  - Guests present at Face to Face

Next Steps

- Refine Documentation
- Transition from Prototype to Reference Implementation
  - Approach NIST Cyber-security Center of Excellence
  - Academia
- Gather Use Cases From Stakeholders
- Actuator Profiles
- Next ‘Hard’ Problem?
- Path To Standardization
  - OASIS
  - IETF
  - ISO
  - Other?
- Future Direction of The OpenC2 Forum

OpenC2 Standardization Timeline



FY Q1 16 | Q2 16 | Q3 16 | Q4 16 | Q1 17 | Q2 17

OpenC2 Definition

- Language Description Document
  - Actions, Syntax
- Use Cases

- **WG Definition Containers**
  - STIX COA
  - *Workflows*
  - Message Fabric
- External Dependencies

Data Models

- *Actuator* (e.g., OpenC2, SACM)
- Target (CyBox)

Gaps in language and data models

Reference Implementations (exercise/stretch the language)

Usage Scenarios

- Pub/Sub, Perimeter Defense
- Software Defined Network
- COTS-based SHORTSTOP
  - Full Mesh, Sensors
  - Internet of Things

GitHub Repository

- Reference Code
- OpenC2 Connectors
- Data Model Encoding and Translation Tools

OpenC2 Connectors

Adoption (vendor and mission buy-in)

- Vendor
  - Orchestrators
  - Actuators

OpenC2 Connectors

Native Support

- Early Adopters

  Cisco Threat INTEL API (CTIA), TCS

Standard (Revs)

Standards Bodies

Socialization and Onboarding

- NIST National CyberSecurity COE (OpenC2 Reference Design)
- Submit Draft Language Description Document to OASIS or other recognized Standards Body

Goals For Today

- Prototype Efforts
  - Capture Lessons Learned
  - Identify and Address Issues
  - Create Reference Implementation
- Implementation Documentation
  - Identify External Dependencies
  - Message Fabric Considerations
- Define Way Forward for Standardization
- Define Future Role of OpenC2 Forum

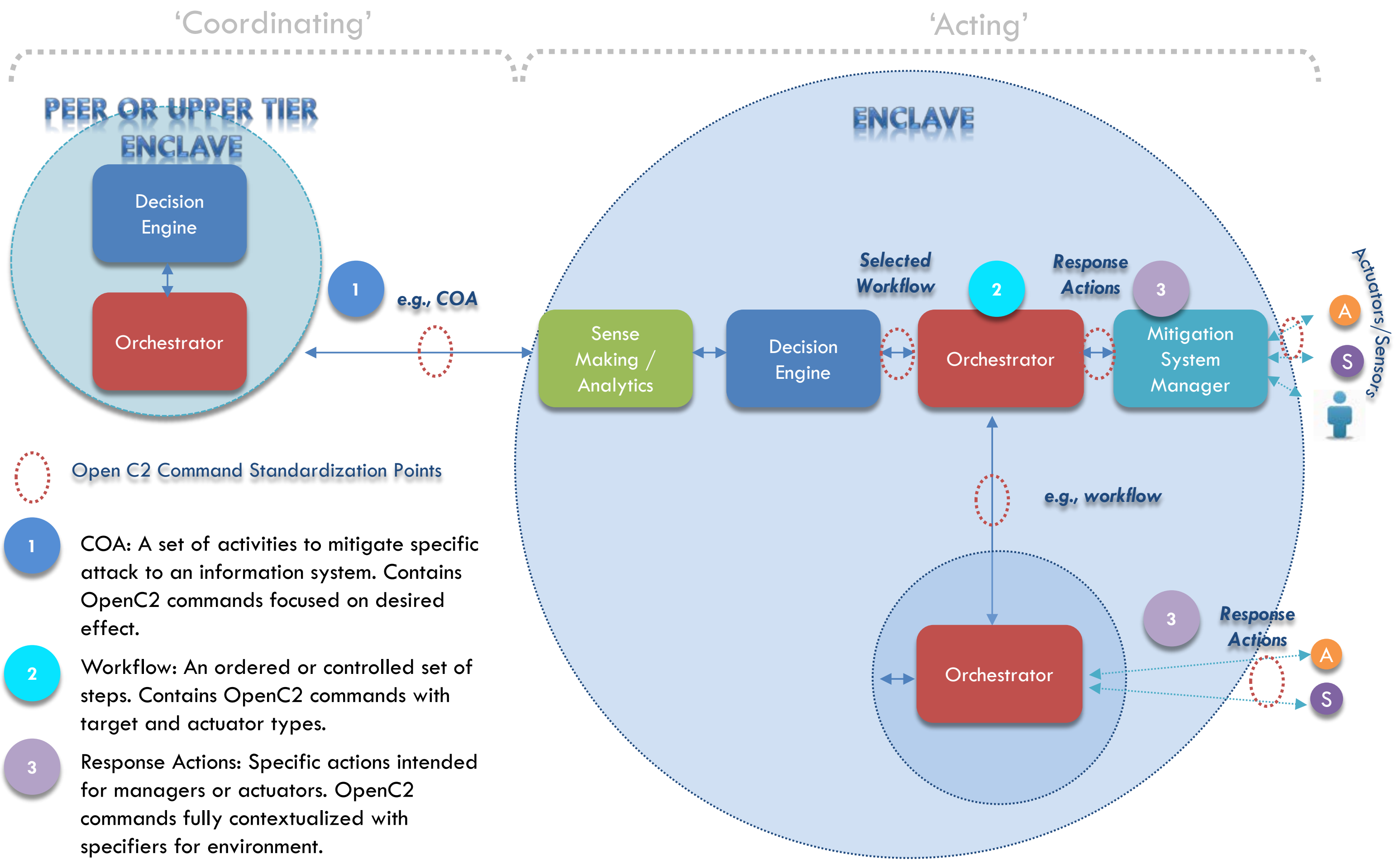
Questions?

Prototype Implementations

Overview

Joyce Fai

OpenC2 Deployment Environments



◌ Open C2 Command Standardization Points

1 COA: A set of activities to mitigate specific attack to an information system. Contains OpenC2 commands focused on desired effect.

2 Workflow: An ordered or controlled set of steps. Contains OpenC2 commands with target and actuator types.

3 Response Actions: Specific actions intended for managers or actuators. OpenC2 commands fully contextualized with specifiers for environment.

OpenC2 Abstract Syntax

```
(
    action = <ACTION_TYPE>,
    target (
        type = <TARGET_TYPE>,
        <target-specifier>
    ),
    actuator (
        type = <ACTUATOR_TYPE>,
        <actuator-specifier>
    ),
    modifiers (
        <list-of-modifiers>
    )
)
```

Documentation in Process

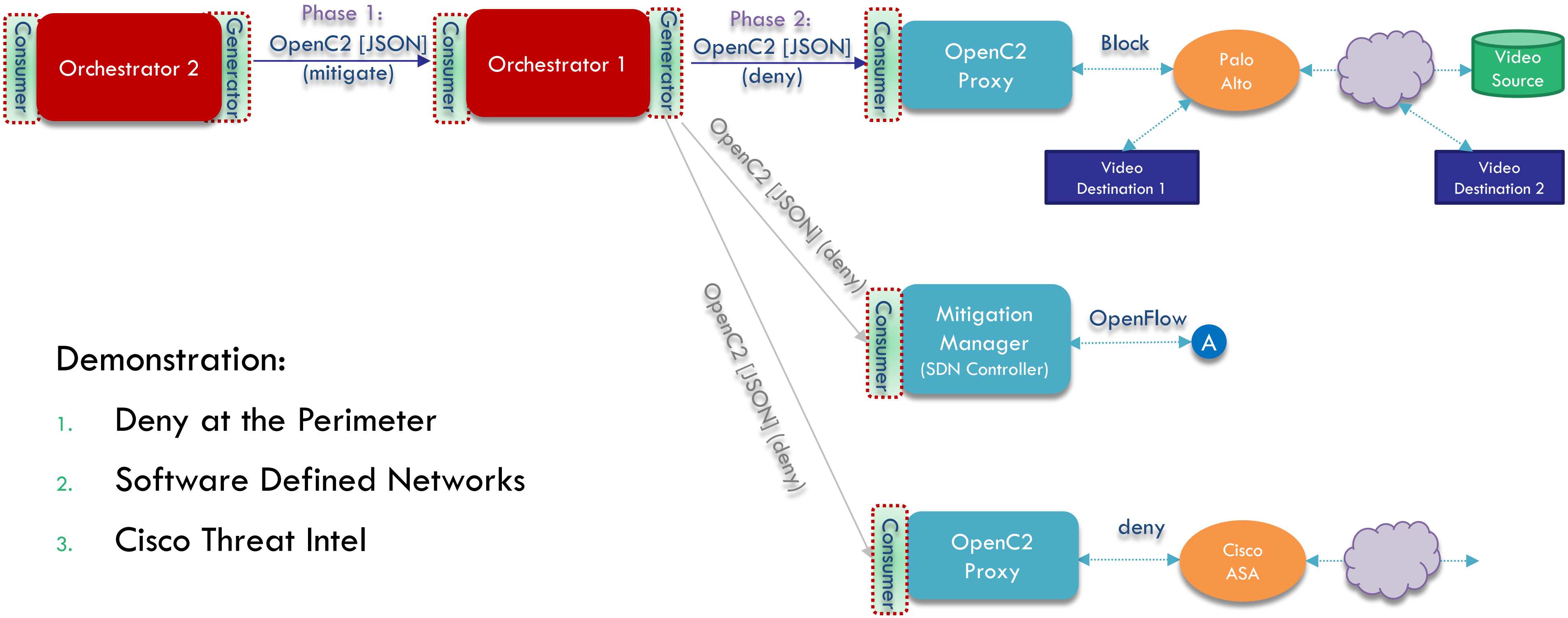
- OpenC2 Language Description Document Ver. 1.0
  - GitHub: `openc2-org/language-description`
  - openc2.org: public link
  - Solicit comments from nonmembers?
  - Comment Resolution circa January 2017?
- IA Considerations Document Ver. 1.0
  - GitHub: `openc2-org/security`
  - openc2.org: public link
  - Comments/ resolution from nonmembers?
- JSON Schema
  - GitHub: `openc2-org/schemas/json`

Prototype Implementations

- Current/ Pending Efforts
  - SDN Controller (Joint SPAWAR, University of Maryland)
    - Subnet Deny, Mitigate, Set, Query
  - Perimeter Firewall (Joint NSA, Phantom Cyber, Cisco)
    - Intra-domain Deny, Mitigate
  - Cisco Threat Intelligence API
    - Cisco ASA interfacing with CTIA
  - NSA/APL Host based Implementation
    - Intra-domain OpenC2 commands
  - OpenC2 and Distributed Network Security Policy Convergence

Update documents based on findings

29 September OpenC2 Face to Face Prototype Implementations - Composite



Demonstration:

1. Deny at the Perimeter
2. Software Defined Networks
3. Cisco Threat Intel

OpenC2 Actions as Multi-methods

Joshua Brule

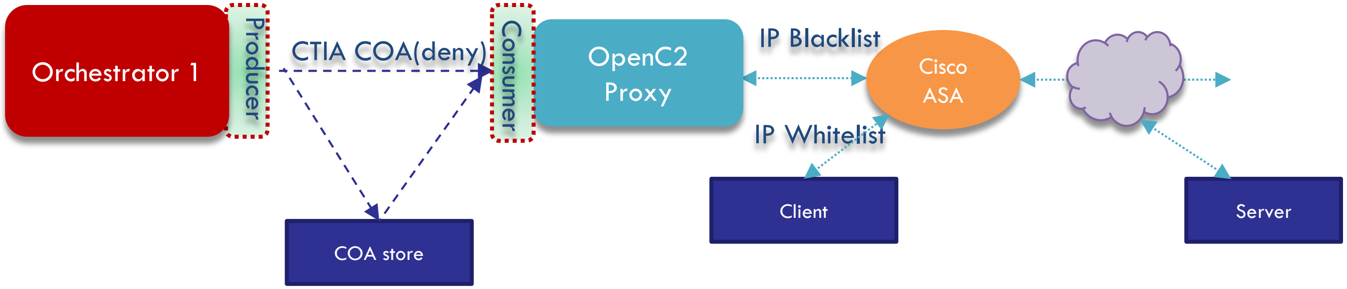
Deny at Perimeter

Larry Salazar

Cisco Threat Intelligence API and OpenC2

Jyoti Verma

Cisco Threat Intelligence API and OpenC2



Demonstration:

- □ Use Cisco Threat Intelligence Service to share actionable threat intel
- □ Based on threat intel, determine that action Send an OpenC2 Deny from Orchestrator 1 Producer to OpenC2 proxy (RESTful API)
- □ OpenC2 proxy consumes Deny and sends CISCO ASA command
- □ OpenC2 Modifier: duration
- □ Deny ACL is programmed on the Cisco ASA for the duration of the command

https://github.com/threatgrid/ctia
https://github.com/threatgrid/ctim/blob/master/src/ctim/schemas/coa.cljc

NSA/APL Host based Implementation

Kevin Miller

OpenC2 and Distributed Network Security Policy Convergence

Eric Voit

OpenC2

SCHEMA DESIGN

Sept 2016

Dave Kemp -- NSA, IA Architectures & Mission Applications

Schema

A schema is:

- *Generic:* a structured framework or plan
- *Database:* the structure of a database system, described in a formal language that defines the tables, the fields in each table, and the relationships between fields and tables
- *XML/JSON:* a description of the elements in a document that can be used to validate each piece of content

A schema can be:

- abstract or concrete
- formal (written in a syntax definition language) or informal

Abstract Syntax*

An Abstract Syntax Language is:

- A formal language for specifying the logical structure of data that is to be exchanged between two endpoints
  - independent of hardware platform, operating system, programming language, local representation, etc.
- Standard sets of rules for encoding instances of logical data structures that are specified in abstract notation
  - for the p

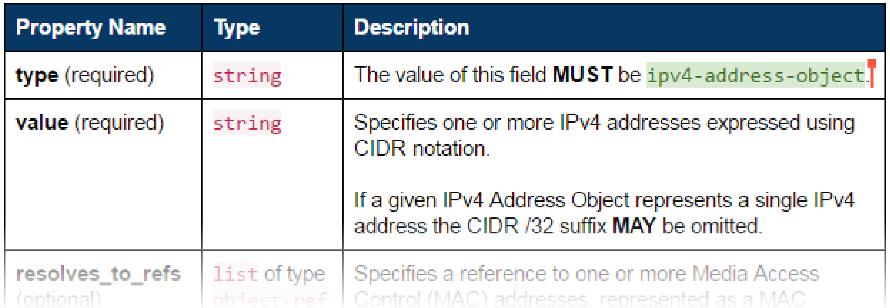
* Description and Principles by Allesandro Trigila
  http://www.ieee802.org/802_tutorials/2010-11/
  Describes ASN.1, but applies to any abstract syntax language

Principles and Benefits* of abstract syntax

- Separation of concerns
  - The description of the logical structure of a message is kept completely separate from the details of the encoding
- Message descriptions are machine-processable
  - This enables the creation and use of software development tools and testing tools that can read and understand the formal definitions
- Encodings are standardized
  - The problem of specifying detailed encodings and the problem of encoding/decoding messages and their fields do not need to be addressed again and again
- Extensibility
  - It is possible to extend a message description in controlled ways while ensuring backward- and forward-compatibility between different version implementations

OASIS CTI Data Definitions

- STIX, TAXII, CybOX are specified in Property Tables
  - **Abstract description**
    - Independent of serialization (XML, JSON, binary)
  - Informal definition of data objects
    - Cannot be machine parsed, validated, or translated
  - Interim step toward formal abstract specifications
    - Would enable automated translation to concrete schemas:
      - XSD
      - JSON Schema
      - Proto3
      - …



Property Name	Type	Description
type (required)	`string`	The value of this field **MUST** be `ipv4-address-object`.
value (required)	`string`	Specifies one or more IPv4 addresses expressed using CIDR notation. If a given IPv4 Address Object represents a single IPv4 address the CIDR /32 suffix **MAY** be omitted.
resolves_to_refs (optional)	`list` of type `object-ref`	Specifies a reference to one or more Media Access Control (MAC) addresses, represented as a MAC

OpenC2 Data Definitions

- Current implementation: Python abstract data structures
  - Executable: enables encoding-independent message validation
    - Defines OpenC2 abstract syntax
      - Unambiguous type checking
    - Validates example messages
    - Supports multiple message formats
      - JSON (multiple dialects)
      - XML
      - Binary
  - Corresponds directly to informal property tables and formal abstract syntax

```
class OpenC2Command(Record):
    vals = [
        ('action', Action, ''),
        ('target', Target, ''),
        ('actuator', Actuator, '?'),
        ('modifiers', Modifiers, '?')]

class Action(Enumerated):
    vals = [
        'scan',        #  1
        'locate',      #  2
        'query',       #  3
         ...
        'remediate']   # 35

class Target(Record):
    vals = [
        ('type', TargetTypeValue , ''),
        ('specifiers', cybox.CyboxObject,
                              '?,{type}')]
```

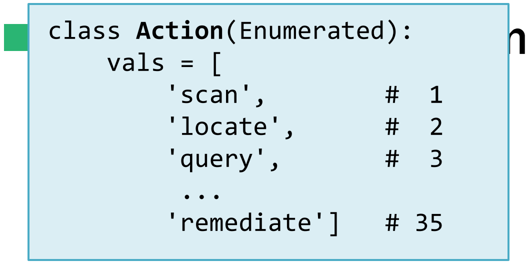
Abstract Syntax - Structures

- JSON Data Model
  - Array – ordered list of items
    - Item has position, no name
  - Object – unordered set of properties
    - Property has name, no position
- Abstract Data Model
  - Record – ordered list of fields
    - Field has both name and position
    - Encoded in JSON as either Array or Object
    - Decoder restores names to Array fields, positions to Object fields
  - Map – unordered set of fields
    - Field has name, no position
    - Encoded in JSON as Object

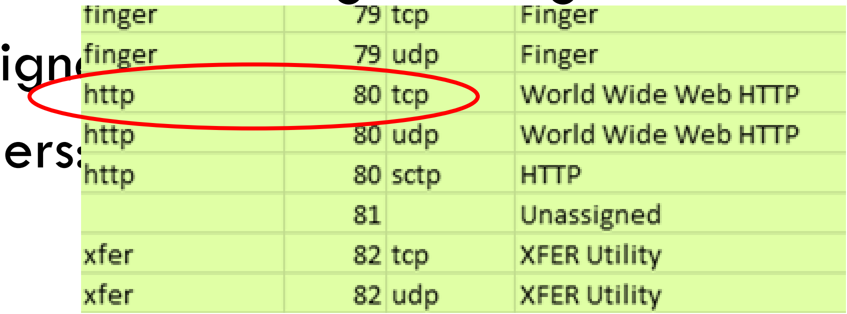
```
class OpenC2Command(Record):
    vals = [
        ('action', Action, ''),
        ('target', Target, ''),
        ('actuator', Actuator, '?'),
        ('modifiers', Modifiers, '?')]
```

Abstract Syntax – Names

- JSON Data Model
  - Names transmitted as strings
    - Field names / property keys (e.g., “type”, “value”, “Action”)
    - Literals in a vocabulary (e.g., “ipv4-address-object”, “TCP”, “scan”)
- Abstract Data Model
  - Names transmitted as either strings or tags
    - Tags (ElementIDs) assign
    - nbers:



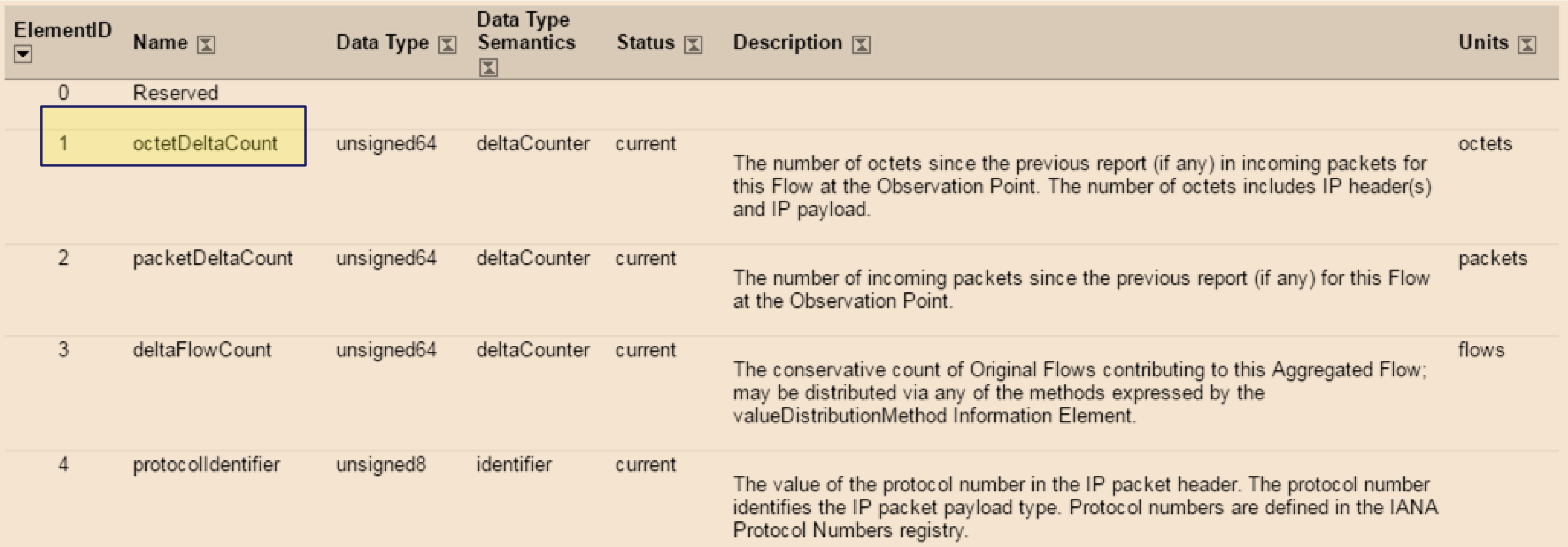
```
class Action(Enumerated):
    vals = [
        'scan',         #  1
        'locate',       #  2
        'query',        #  3
         ...
        'remediate']    # 35
```



finger	79 tcp	Finger
finger	79 udp	Finger
http	80 tcp	World Wide Web HTTP
http	80 udp	World Wide Web HTTP
http	80 sctp	HTTP
	81	Unassigned
xfer	82 tcp	XFER Utility
xfer	82 udp	XFER Utility

Abstract Syntax – Names (cont.)

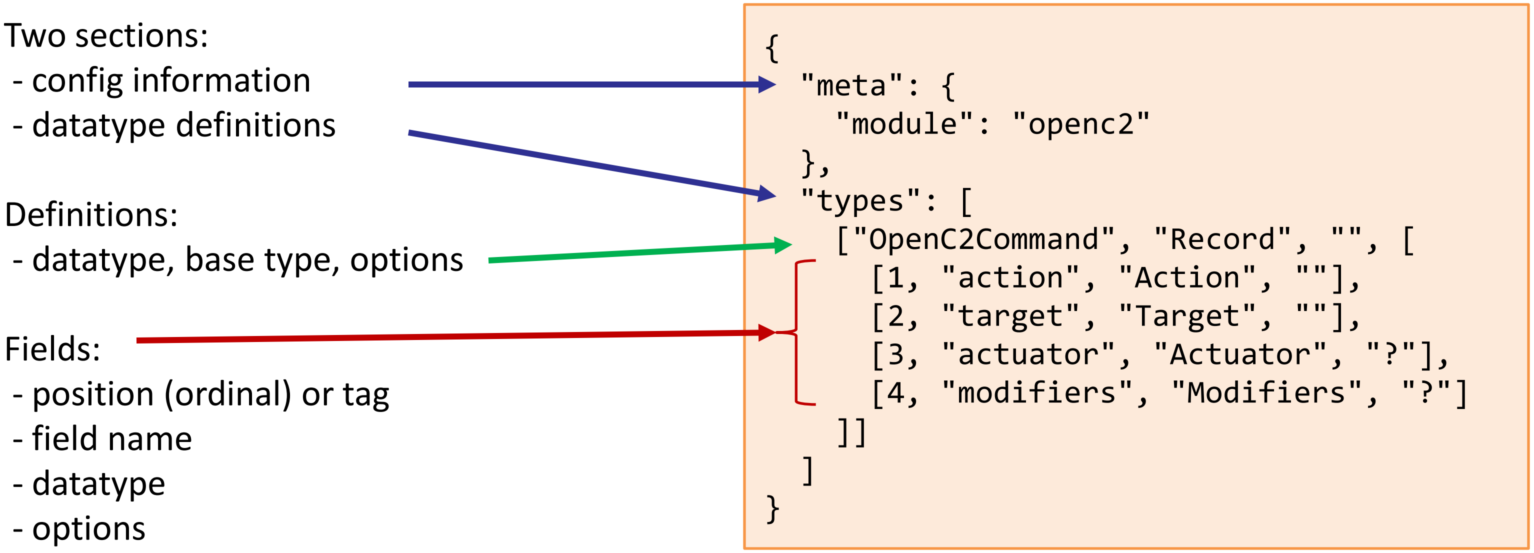
- IP Flow Information Export (IPFIX) element registry
  - Abstract syntax assigns IDs (1) to names (“octetDeltaCount”)
  - Concrete encoding uses one or the other
  - Decoder supplies name corresponding to received ID
  - Namespace used to identify registry



ElementID	Name	Data Type	Data Type Semantics	Status	Description	Units
0	Reserved					
1	octetDeltaCount	unsigned64	deltaCounter	current	The number of octets since the previous report (if any) in incoming packets for this Flow at the Observation Point. The number of octets includes IP header(s) and IP payload.	octets
2	packetDeltaCount	unsigned64	deltaCounter	current	The number of incoming packets since the previous report (if any) for this Flow at the Observation Point.	packets
3	deltaFlowCount	unsigned64	deltaCounter	current	The conservative count of Original Flows contributing to this Aggregated Flow; may be distributed via any of the methods expressed by the valueDistributionMethod Information Element.	flows
4	protocolIdentifier	unsigned8	identifier	current	The value of the protocol number in the IP packet header. The protocol number identifies the IP packet payload type. Protocol numbers are defined in the IANA Protocol Numbers registry.	

JSON Abstract Syntax Notation (JASN)

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



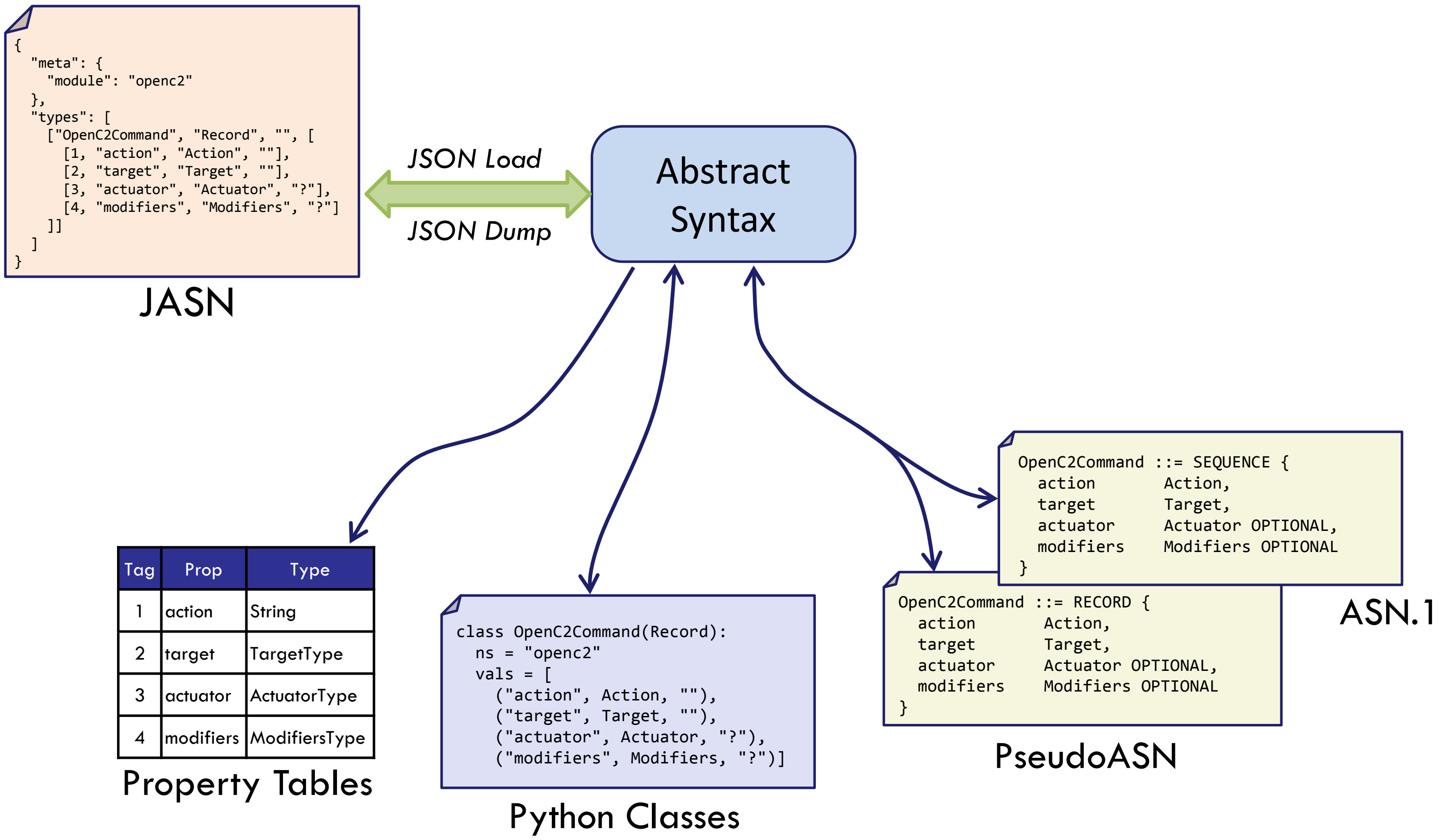
Two sections:
- config information
- datatype definitions

Definitions:
- datatype, base type, options

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

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

Abstract Schema Representations



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

JASN

Tag	Prop	Type
1	action	String
2	target	TargetType
3	actuator	ActuatorType
4	modifiers	ModifiersType

Property Tables

```
class OpenC2Command(Record):
  ns = "openc2"
  vals = [
    ("action", Action, ""),
    ("target", Target, ""),
    ("actuator", Actuator, "?"),
    ("modifiers", Modifiers, "?")]
```

Python Classes

```
OpenC2Command ::= SEQUENCE {
  action        Action,
  target        Target,
  actuator      Actuator OPTIONAL,
  modifiers     Modifiers OPTIONAL
}
```

ASN.1

```
OpenC2Command ::= RECORD {
  action        Action,
  target        Target,
  actuator      Actuator OPTIONAL,
  modifiers     Modifiers OPTIONAL
}
```

PseudoASN

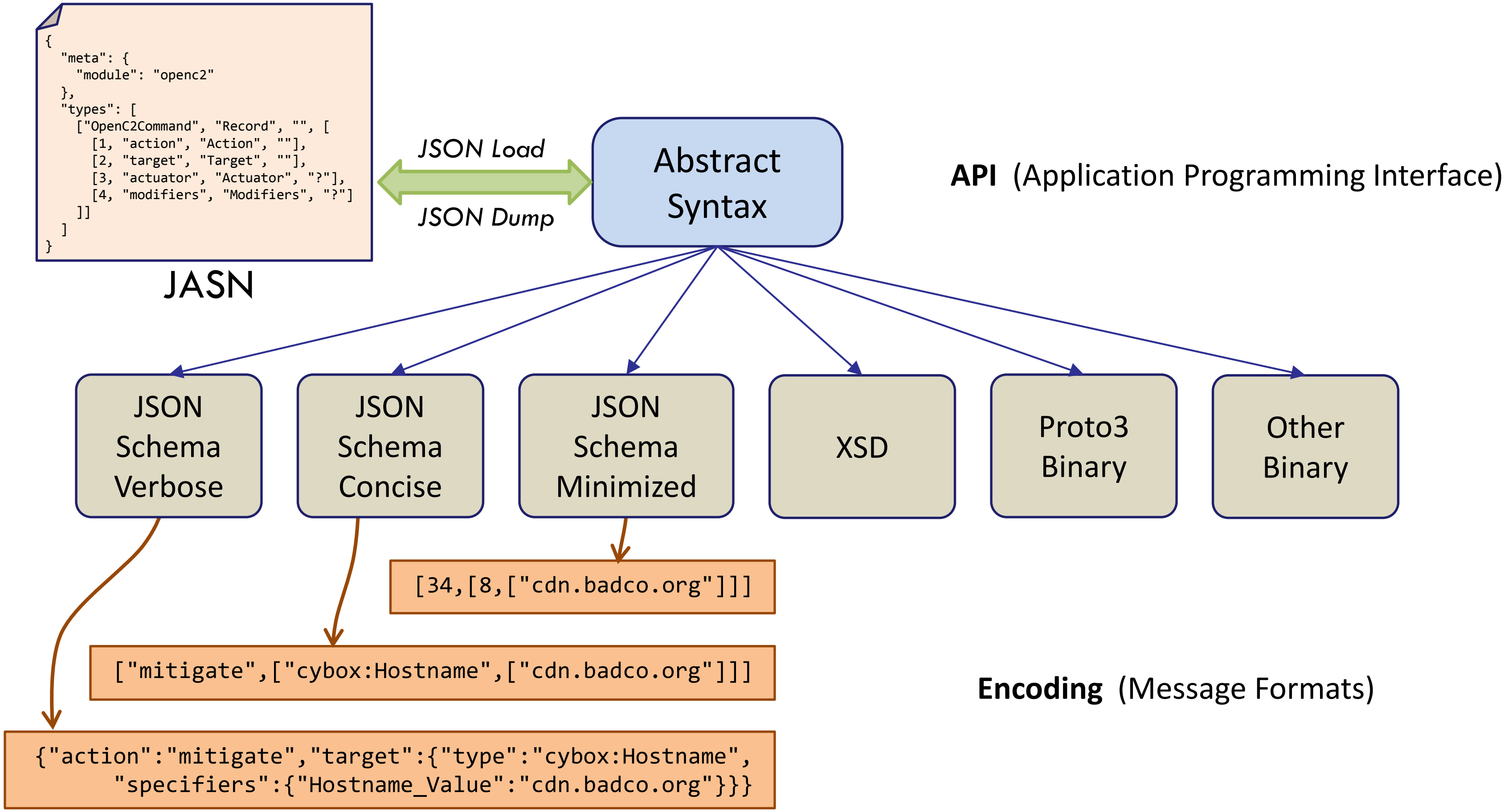
Pseudo-ASN (PASN)

- Mostly ASN.1, but modified for ease of use
  - ASN.1 has no first-class map (key:value pair) type
    - SEQUENCE / SEQUENCE OF and SET / SET OF are the only compound ASN.1 types
    - Table Constraint syntax is general but cumbersome
    - PASN defines MAP to represent Identifier:Typereference pairs
  - ASN.1 restricts case for Identifier and Typereference
    - PASN allows both upper and lower case first character
  - ASN.1 SEQUENCE does not support encoding modes
    - JSON Encoding Rules (to be defined) might add encoding modes
    - PASN defines RECORD to be encoded as either JSON object or array
  - PASN requires explicit tags for names
    - RECORD field tags are optional, must be ordinal if present
  - PASN does not allow anonymous type definitions
    - Fields contain references to named types
    - Supports direct translation to JASN without compiling

JSON Encoding Modes

- Verbose
  - RECORD encoded as Object
  - Highest bandwidth
  - Arguably most human-readable (explicit field names)
- Concise
  - RECORD encoded as Array
  - Reduced bandwidth
  - Arguably more readable (no field name clutter)
- Minimized
  - RECORD encoded as Array, Names encoded as Tags
  - Most bandwidth efficient, least readable
  - Use directly for transmission, or as visualization of binary encoding

Concrete Schema Generation



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

JASN

JSON Load

JSON Dump

Abstract Syntax

API (Application Programming Interface)

JSON Schema Verbose

JSON Schema Concise

JSON Schema Minimized

XSD

Proto3 Binary

Other Binary

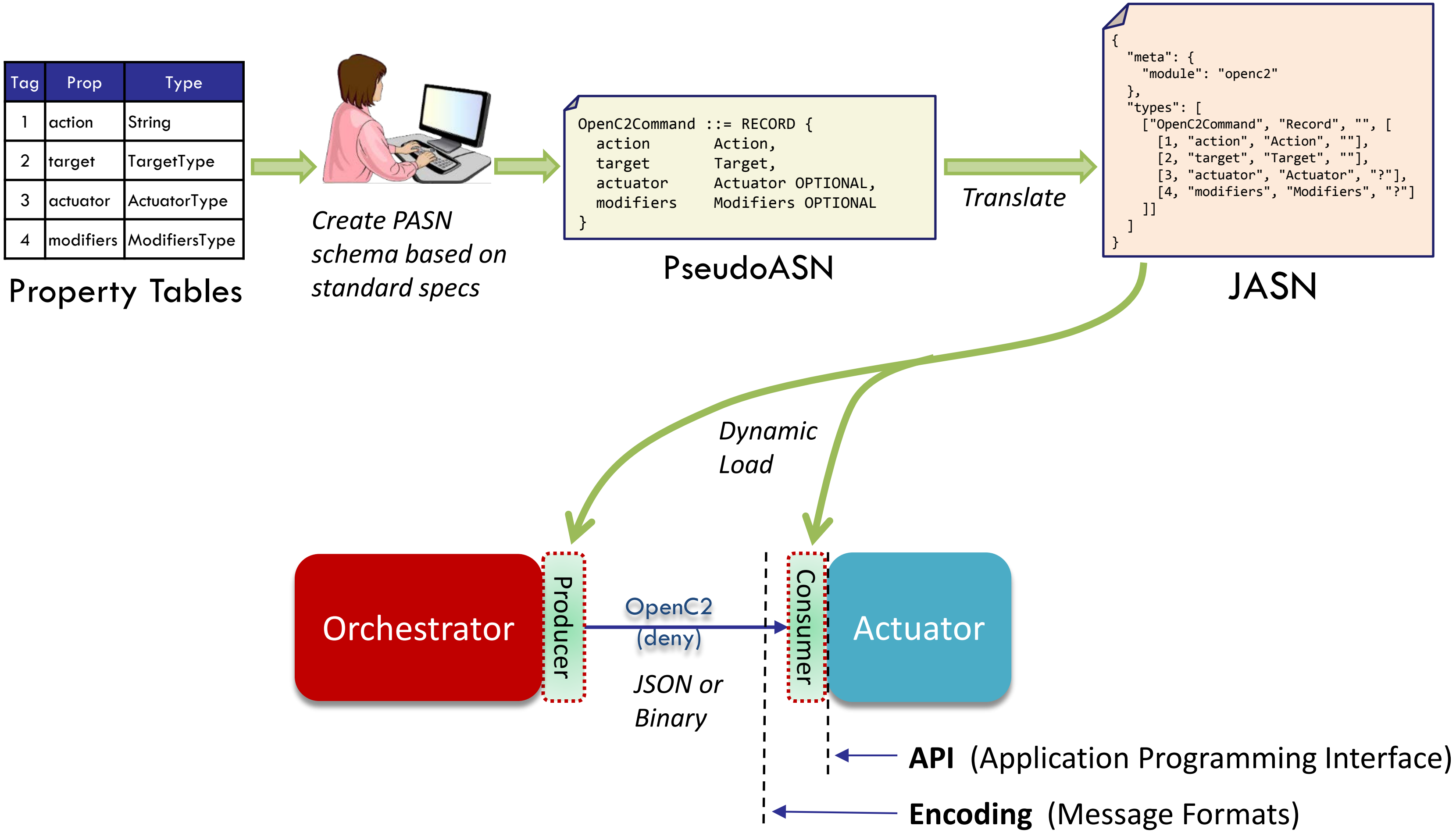
```
[34,[8,["cdn.badco.org"]]]
```

```
["mitigate",["cybox:Hostname",["cdn.badco.org"]]]
```

```
{"action":"mitigate","target":{"type":"cybox:Hostname",
     "specifiers":{"Hostname_Value":"cdn.badco.org"}}}
```

Encoding (Message Formats)

Application Development



Tag	Prop	Type
1	action	String
2	target	TargetType
3	actuator	ActuatorType
4	modifiers	ModifiersType

Property Tables

Create PASN schema based on standard specs

```
OpenC2Command ::= RECORD {
  action        Action,
  target        Target,
  actuator      Actuator OPTIONAL,
  modifiers     Modifiers OPTIONAL
}
```

PseudoASN

Translate

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

JASN

Dynamic Load

JSON or Binary

API (Application Programming Interface)

Encoding (Message Formats)

Message Structure

- API supports structure (nested objects) and template (object with path keys) formats
  - Identical information, lossless bidirectional conversion
  - Template format may be easier for applications to work with

```
{ "ACTION": "DENY",
  "TARGET": {
    "type": "cybox:Network_Connection",
    "specifiers": {
      "Layer3Protocol": "IPv4",
      "Layer4Protocol": "TCP",
      "SourceSocketAddress": {
        "IP_Address": {
          "Address_Value": "any"}},
      "DestinationSocketAddress": {
        "IP_Address": {
          "Address_Value": "10.10.10.2"}}}},
  "ACTUATOR": {
    "type": "network-firewall",
    "specifiers": {
      "asset_id": "30"}},
  "MODIFIERS": {
    "context_ref": 91}
}
```

```
{ "ACTION": "DENY",
  "TARGET.type": "cybox:Network_Connection",
  "TARGET.specifiers.Layer3Protocol": "IPv4",
  "TARGET.specifiers.Layer4Protocol": "TCP",
  "TARGET.specifiers.SourceSocketAddress.IP_Address.Address_Value": "any",
  "TARGET.specifiers.DestinationSocketAddress.IP_Address.Address_Value":
"10.10.10.2",
  "ACTUATOR.type": "network-firewall",
  "ACTUATOR.specifiers.asset_id": "30",
  "MODIFIERS.context_ref": 91
}
```

Lessons Learned

- Distributed assignment (namespaces):
  - No generally-accepted namespace approach for JSON
    - Forced to roll our own to use both CybOX 2 and CybOX 3
    - Need standardized namespace approach
- Balance between nesting and referencing:
  - STIX 1 allowed structures with unlimited nesting levels
  - STIX 2 (and CybOX 3) forbid nesting entirely
    - Result: IP Address object uses a reference (pointer) to a MAC Address object, Excessive message overhead for containers, References complicate message definition and validation
    - Everything in moderation – allow 1-2 nesting levels, but not unlimited

Lessons Learned (cont.)

- Think Abstract!
  - Designers need names to understand data
    - Result: Protocols defined at transport level send names, wasting bandwidth
    - Design at abstract level (write message APIs in "source code")
      - Communicate using efficient concrete data and schemas ("machine code")
      - Avoid mistakes like using objects to emulate arrays
    - Message templates can ease application integration
  - Tools can use abstract schema
    - Menu-based or template-based message composers
    - "Wireshark" display module
  - Applications should provide mechanism, not policy
    - Producer selects message encoding at runtime
    - Consumer config'ed to accept one or many message encodings
    - Strict (lint) or permissive (case-insensitive) receive modes

Next Steps

- Initial Python codec release
  - Current code is incomplete, alpha level proof-of-concept
    - Need JASN-based decoder
    - Need encoder methods
    - Need CybOX 3 JASN definitions
    - Need concrete schema generator
    - Need test suite
    - Need documentation
    - Need …
- Socialize abstract design approach with CTI

Lessons and Implementation Considerations

Group Discussion

Summary

- □ JSON makes Python development easier
- □ OpenC2/Native command mapping not always 1:1
- □ Some commands need more detail, specific modifiers, explicitly defined default values
- □ Limited definition for handling responses to commands
- □ Implementation Consideration: REST Interfaces
- □ Need standardized namespace approach

From: Cisco Threat INTEL API

- Example usages
- Change in Language Description
  - Response Action need to be more well defined
  - Need to reserve a standard port number for command and control
  - Device implement REST interfaces
- Implementation Trades (e.g., schema)

From NSA/APL: Verbosity

CybOX, XML

- Originally agreed to flexibility in design
- Eschewed
  - XML in favor of JSON
  - Aligning field names with CybOX object names
- JSON made the Python library significantly easier to develop
- Using fields:
  - Significantly cut down on CybOX verbosity
  - Added flexibility for different transports
  - Easily mapped intent back to CybOX without using it

From NSA/APL: Flexibility

CarbonBlack Example

- Started implementing by mapping to CarbonBlack’s REST API
- Migrated to using its Python API
- OpenC2 commands did not change
  - Mappings are not always 1:1
  - Currently requires a robust, open, well documented API

From NSA/APL: Intent

Interpretation of OpenC2 Commands

- Some commands need more detail or require explicitly defined default values
- Consider blocking an IP at a firewall
  - Given currently required elements of OpenC2, how?
    - Silently drop packets
    - Block with icmp
- Bottom line: some commands either need to require specific modifiers or need explicitly defined default behaviors

From NSA/APL: Responses

Limited definition for handling responses to OpenC2 commands

- Originally ignored RESPONSE due to lack of discussion in working group
- Even as specified, combinations of responses need to be considered:
  - Where to redirect output of executed command (acknowledging INVESTIGATE’s modifier “report-to”)
  - Where to respond with status of command
- Effectively, how should we nest transport-agnostic (message bus queue name?) response requirements (status and output to different places) into one command?

From NSA/APL: Responses

Limited definition for handling responses to OpenC2 commands

- How should we handle uniquely identifying OpenC2 commands?
  - Metadata wrapper object
  - Packet header
- We used:
  - Metadata JSON wrapper with OpenC2 command nested in
  - Added UUID and more abstract reply-to fields
- Implemented actions/responses with a message fabric
  - Commands go out on a topic logically separating classes of actuators
  - Response queue name aligned with the UUID of the command

From NSA/APL: Responses

Limited definition for handling responses to OpenC2 commands

- OpenC2 responses…
  - Where (already covered)
  - <u>How?</u>
- RI: redirect output from an API response back to the orchestrator
  - Standardizing response payload will be a significant challenge across the vendor space
  - Leaves text or object parsing up to the orchestrator
  - Might be out of scope for the OpenC2 language, but worth noting nonetheless

Implementing Structured COA for STIX

Jyoti Verma, Cisco

Bret Jordan, BlueCoat

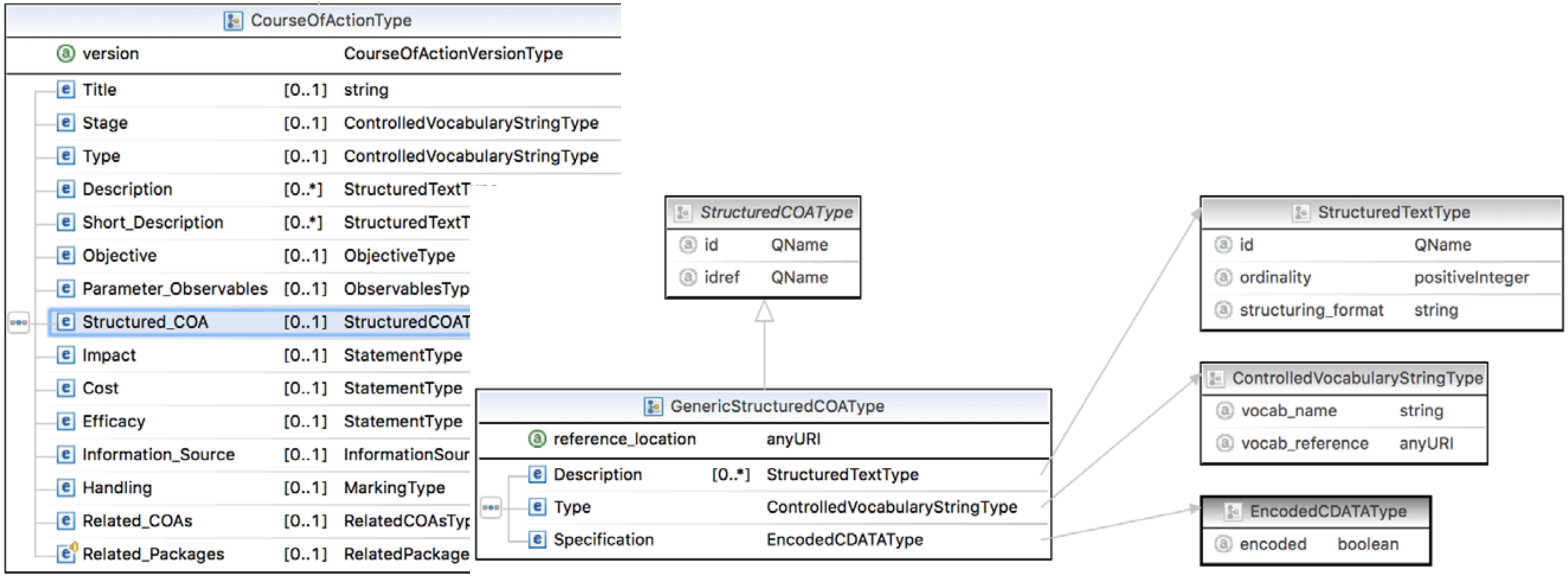
OpenC2 and STIX

- Thanks to all members of the OpenC2 STIX subgroup specifically Dave, Jason, Joyce, Joe, without whom we couldn't have come this far!!
- Special thanks to Bret Jordan, the co-chair of STIX for supporting us!

- What we did so far
  - Worked with the main OpenC2 group to refine the JSON schema
  - Discussed STIX and Cybox constructs and how OpenC2 might be carried in the STIX envelope
  - Created a proposal for STIX 1.2
  - Created a working proposal for STIX 2.0 here
- What's next
  - Define OpenC2 namespace for targets
  - Define vocabularies for action types and actuators
  - Work through concepts of STIX 2.0 such as patterning, relationships etc. that might affect the representation of OpenC2 in STIX

How to use OpenC2 with STIX today?

- □ STIX 1.x – embed the OpenC2 message in the Specification field as a String



CourseOfActionType		
version		CourseOfActionVersionType
Title	[0..1]	string
Stage	[0..1]	ControlledVocabularyStringType
Type	[0..1]	ControlledVocabularyStringType
Description	[0..*]	StructuredTextT
Short_Description	[0..*]	StructuredTextT
Objective	[0..1]	ObjectiveType
Parameter_Observables	[0..1]	ObservablesTyp
Structured_COA	[0..1]	StructuredCOAT
Impact	[0..1]	StatementType
Cost	[0..1]	StatementType
Efficacy	[0..1]	StatementType
Information_Source	[0..1]	InformationSour
Handling	[0..1]	MarkingType
Related_COAs	[0..1]	RelatedCOAsTyp
Related_Packages	[0..1]	RelatedPackage

StructuredCOAType	
id	QName
idref	QName

GenericStructuredCOAType		
reference_location		anyURI
Description	[0..*]	StructuredTextType
Type		ControlledVocabularyStringType
Specification		EncodedCDATAType

StructuredTextType	
id	QName
ordinality	positiveInteger
structuring_format	string

ControlledVocabularyStringType	
vocab_name	string
vocab_reference	anyURI

EncodedCDATAType	
encoded	boolean

- □ STIX 2.0 – Use a custom object for OpenC2 and use in the course_of_action object

Call to Arms

- We need your help to make OpenC2 supported by STIX
- If you are already a member
  - Please voice your opinion on email and slack channels
- To become a member or join slack …

What is the next REALLY Hard Problem

Group Discussion

The Next Really Hard Problem

- Actuator OpenC2 Profiles
- Implementation Considerations
- Reference Implementations
- Define External Dependencies
- Message Fabric
- Engage Standards Bodies

BLUF: Articulate Six Month Roadmap

The Next Really Hard Problems?

- Actuator OpenC2 Profiles
  - SDN Profile in Draft
  - Expand Use Cases
  - Other Actuator Types?
- Implementation Considerations Document?
  - Sequence Numbering, Acknowledgement, Responses
  - Capture Lessons Learned; schemas
- Reference Implementations Binary Encoding?
    - IoT Prototype Implementation?
  - Alert/ Response
- Which Standards Body?
  - Leverage Pre-existing Relationships (e.g., STIX)
  - Determine what standards body (e.g., OASIS) will require to stand up a subcommittee
- Message Fabric?

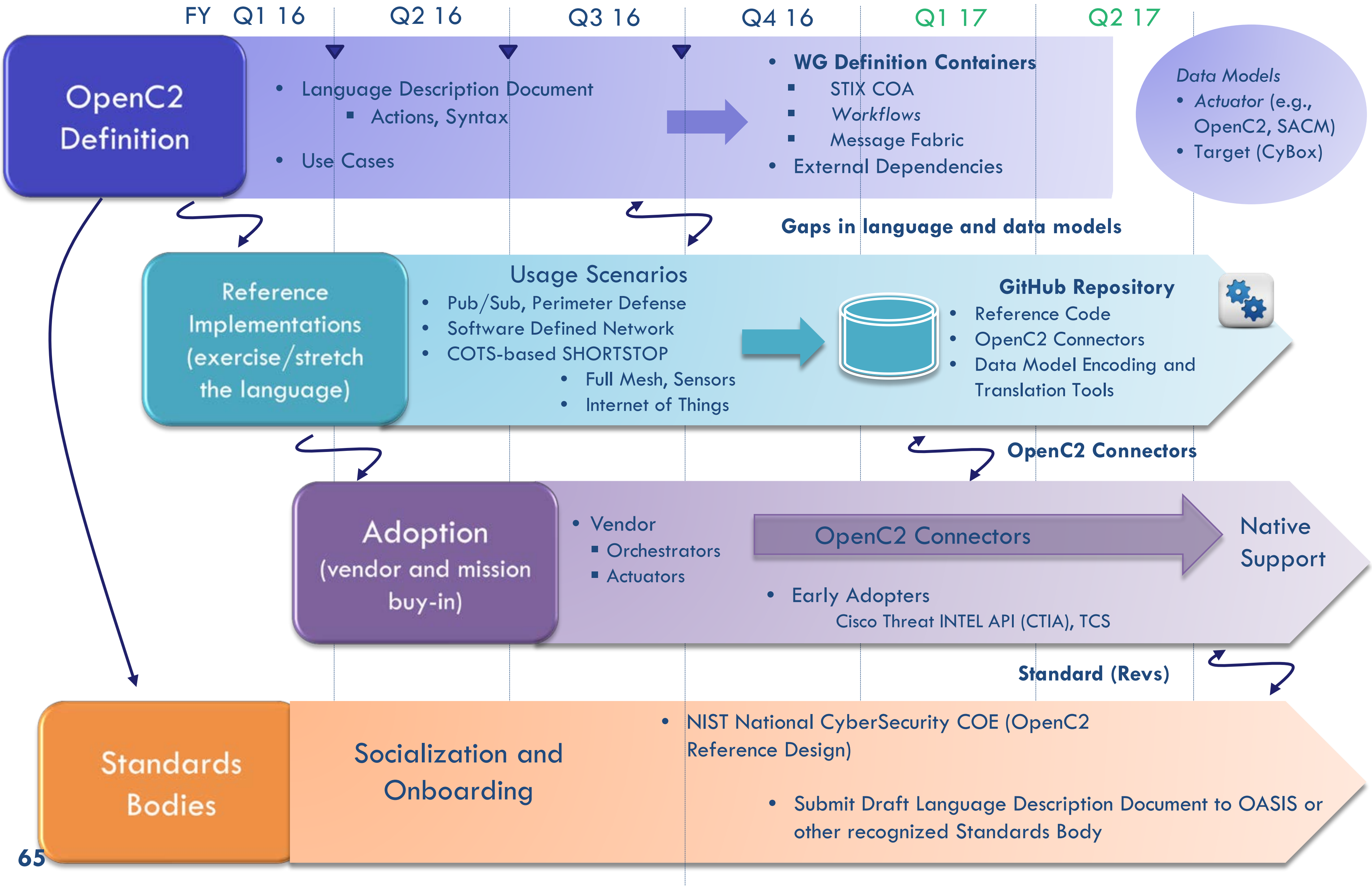
OASIS Overview

Bret Jordan, BlueCoat

Path to Standardization

Group Discussion

OpenC2 Standardization Timeline



FY Q1 16 | Q2 16 | Q3 16 | Q4 16 | Q1 17 | Q2 17

OpenC2 Definition

- Language Description Document
  - Actions, Syntax
- Use Cases

- **WG Definition Containers**
  - STIX COA
  - *Workflows*
  - Message Fabric
- External Dependencies

Data Models

- *Actuator* (e.g., OpenC2, SACM)
- Target (CyBox)

Gaps in language and data models

Reference Implementations (exercise/stretch the language)

Usage Scenarios

- Pub/Sub, Perimeter Defense
- Software Defined Network
- COTS-based SHORTSTOP
  - Full Mesh, Sensors
  - Internet of Things

GitHub Repository

- Reference Code
- OpenC2 Connectors
- Data Model Encoding and Translation Tools

OpenC2 Connectors

Adoption (vendor and mission buy-in)

- Vendor
  - Orchestrators
  - Actuators

OpenC2 Connectors

Native Support

- Early Adopters

  Cisco Threat INTEL API (CTIA), TCS

Standard (Revs)

Standards Bodies

Socialization and Onboarding

- NIST National CyberSecurity COE (OpenC2 Reference Design)

- Submit Draft Language Description Document to OASIS or other recognized Standards Body

Documentation in Process

- OpenC2 Language Description Document Ver. 1.0
  - Currently posted on Public Facing Website
  - Solicit comments from nonmembers
  - Comment Resolution circa January 2017
- IA Considerations Document Ver. 1.0
  - Currently posted on Public Facing Website
  - Comments/ resolution from nonmembers?
- Schema
  - JSON Schema Currently posted on GitHub
  - ASN.1 in Progress on Private GitHub Repository

Data Modeling/ STIX

- STIX COA WG
  - OpenC2 to be included in STIX 2.XJan 2017
  - Cybox3.0 Data Model for STIX 2.0 Structured COA Field
- Data Modeling
  - Flat Data Model
    - JSON Schema in process (pending)
    - ASN.1 Model in process (circa Feb. 2017)
  - YANG Model
- Actuator Data Modeling Issues

Transition to Reference Implementations

- Summarize Lessons Learned
- Expand Use Cases
- Make Prototype Implementations Readily Available
- Address External Dependencies in Reference Implementations (e.g., message fabric, IA considerations)
- Incorporate Findings Into the Documentation
- Outreach to NCCOE?

Many Thanks to the Members for their Contributions to the Prototyping Efforts

Way Forward for OpenC2

Group Discussion

Questions?

Comments?

Complaints?