Interface to Security Functions: An overview and comparison of I2NSF and OpenC2

Matteo Repetto

***Abstract*—Recent management paradigms for software-defined infrastructures bring more agility in the creation and operation of digital services, but also introduce new cyber-security issues due to fast-changing environments, dynamic topologies, and wider attack surfaces. Rigid and statically-configured architectures are no more suitable for the detection of cyber-attacks in mixed cloud/6G/IoT environments, hence new frameworks must be designed that are more flexible and adaptable, up to become cognitive. A fundamental step in this direction is represented by the adoption of common interfaces to orchestrate heterogeneous and multi-vendor security functions in an homogeneous way.**

In this paper, we consider two recent interfaces to security functions that are representative of different approaches and industrial domains, namely I2NSF and OpenC2. We briefly review the latest advances in their definition, provide a deep comparison and outline major limitations and research challenges for concrete application scenarios. The main purpose of our work is to make an unbiased evaluation of the current status of these standards and to foster researchers to actively contribute to their development by adopting them and by proposing further extensions and refinements.

***Index Terms*—Cyber-defense, security orchestration and automation, interfaces, security functions**

I. Introduction

The evolution of 5G networks into a large, pervasive, and powerful computing continuum is radically changing the way digital services are created and operated. Service-oriented architectures and software-defined infrastructures boost new management models, where digital services are composed by chaining software-defined resources and functions from heterogeneous domains: Network Function Virtualization (NFV), cloud/edge/fog applications, Internet of Things (IoT), and data [1], [2]. They also continuously evolve at run-time, according to human-defined orchestration rules or, in perspective, even cognitive Artificial Intelligence (AI) processes. Effective protection of such systems against cyber-threats becomes more challenging, due to the lack of a sharp and effective security perimeter, the usage of third parties' infrastructures and services, and the introduction of serverless computing. A transition towards adaptive and agile cyber-defense architectures is therefore necessary beyond existing models, where most cyber-security appliances are manually configured for static environments and work in isolation [3], which eventually delays mitigation and response actions. The ultimate goal is homogeneous, seamless, and transparent orchestration of capillary and programmable monitoring, detection, and enforcement capabilities at the edge and in user devices [1].

The realization of adaptive and agile cyber-defense frameworks requires knowledge of the service composition and topology at run-time, including security functions that implement monitoring, detection, and enforcement tasks. Relevant examples of this approach are already available for individual and homogeneous domains [4], [5], but extension to a more general scenario relies on common interfaces for exposing, discovering, and controlling security capabilities embedded in digital infrastructures and services.

Standard interfaces to security functions would simplify the composition of monitoring, detection, and enforcement processes over heterogeneous service chains deployed in mixed 5G/6G/cloud/IoT infrastructures, removing the need for inflexible and hardly manageable adapters, as commonly happens with Cloud Access Security Brokers (CASBs). In this respect, Interface to Network Security Functions (I2NSF) [6] and Open Control and Command (OpenC2) [3] were proposed by IETF and OASIS, respectively. They address the same problem, but from a different perspective and with different use cases in mind, which reflect the peculiarities of the corresponding communities, namely the telecommunication industry for IETF and software industry for OASIS. The selection of these standards is motivated by the fact that they are the first and only management interfaces explicitly designed for cyber-security operations that are backed by significant communities.

The concrete contribution of this paper is twofold:

- a short review of the current status of I2NSF and OpenC2, including updates with respect to previous works;
- a comparative analysis of these interfaces, together with the identification of existing gaps, application scenarios, and future research directions.

The rest of the paper is organized as follows. First, we give a brief overview of Related Work. Then, we describe I2NSF and OpenC2. Afterwards, we compare the two initiatives, and outline main limitations and research directions for target application scenarios. Finally, we give our conclusion.

II. Related work

The geographical distribution of cloud infrastructures has always been seen as an opportunity to fulfill conflicting requirements in terms of processing capabilities, latency, reliability and availability [7]. Cui *et al*. [2] also consider telecommunication networks, by discussing the implementation of Service

M. Repetto is with the Institute for Applied Mathematics and Information Technologies "E. Magenes" (IMATI), National Research Council (CNR), Italy. Email: matteo.repetto@cnr.it.

Function Chaining (SFC) over a federated environment made of cloud and edge installations. Repetto *et al.* [1] further extend the scenario, by including the IoT and data stores; the same paper also points out the need to monitor both the service itself and the underlying infrastructure, to address the widest threat landscape.

The need to discover and configure remote security agents in large computing environments has been a recurring issue. In the SECURED project [5], high-level policies are translated into specific configurations, similarly to CASBs; unfortunately, this approach does not scale well with heterogeneous interfaces, because a different "adapter" or "driver" should be provided for each of them. A similar approach was followed by Bondan at al. [8], who use an orchestrator abstraction driver to retrieve information about running Virtual Network Functions (VNFs) and to notify anomalies; however, the scope is limited to SFC status and network configuration, hence leaving out any form of integrity verification and network attack. A similar approach is followed by Repetto *et al.* [4], who also introduce an abstraction layer to discover and configure capabilities of security agents. Morais *et al.* [9] describe how Manufacturer Usage Description (MUD) can be matched with a Malicious Traffic Description (MTD) data model to mitigate vulnerabilities in an IoT installation. Even if MUD does not include security capabilities, it anyway represents a minimal example of interface to retrieve context information. Carrega *et al.* [10] implemented the GUARD framework for remote management of cyber-security functions and dynamic composition of monitoring, detection, and response processes for digital service chains; in this case, a Smart Data Model (SDM) is used to discover their location, to describe their capabilities, and to configure them.

The I2NSF framework was previously described by Hyun *et al.* [6], together with its integration with a Software-Defined Networking (SDN) controller for Voice-over-IP (VoIP) and time-dependent web access control use cases. An overview of OpenC2 was originally given by Mavroeidis and Brule [3], but without including comparison with alternative approaches (i.e., I2NSF). In both cases, the authors were directly involved in the definition of the standards, so an unbiased discussion of their limitations and technical gaps is still missing.

III. Interface to Network Security Functions

The I2NSF framework introduces a set of common interfaces to automatically discover the capabilities of Network Security Functions (NSFs), to derive behavioral rules from high-level policies, to automate response to external triggers, and to monitor their behavior and integrity over time. Relevant use cases include content filtering in access networks, on-demand distributed firewalling, attack detection and mitigation [11].

The current architecture of the I2NSF framework [12] is depicted in Fig. 1. In Fig. 1(a), the I2NSF functional elements are shown, together with the interfaces between them. In Fig. 1(b), the high-level concept for I2NSF is depicted, namely the translation between the high-level policies into concrete configuration rules; the proposed workflow to implement this translation is then shown in Fig. 1(c). I2NSF data models for capabilities and configurations of NSFs are described in several IETF working drafts.[1]

A. I2NSF functional elements

The I2NSF architecture includes the necessary elements to implement closed-loop response to network-based attacks through re-configuration of NSFs.

1) I2NSF user: This element includes any potential source of high-level security policies (*intents*). Besides humans, this definition also accounts for applications (e.g., video-conference network manager), and network/IoT management systems that define policies in NSF-agnostic way.

2) Network Security Function: This represents a monitoring and/or enforcement function, such as firewall, Intrusion Detection System (IDS)/Intrusion Protection System (IPS), antivirus, Virtual Private Network (VPN), and so on. Though not strictly necessary, the use of VNFs simplifies automation of their life-cycle management.

3) Security Controller: The Security Controller (SC) is the smart component that generates configuration rules from intents (see Fig. 1(b)). This operation includes the selection of the most appropriate types of NSFs to cooperatively enforce the policy, the generation of low-level rules, and finally the configuration of each concrete instance. Additionally, the SC is also responsible for registration of NSF capabilities.

4) Developer's Management System: The Developer's Management System (DMS) is a registry that describes the NSFs of a specific vendor and their capabilities. In general, there can be multiple DMSs available to the SC, to take advantage of vendor diversity in terms of capabilities, performance and cost.

5) Analyzer: This is the smart engine that derives new behavioral policies from events and data generated by NSFs (perhaps using some form of AI), hence automating mitigation of and response to cyber-attacks.

B. I2NSF interfaces

I2NSF interfaces are defined in terms of YANG[2] modules, which are then encoded in XML[3] to be transferred over RESTCONF or NETCONF. Beside actual data, I2NSF modules also define message types, namely queries and notifications. Security and access controls are demanded to RESTCONF/NETCONF.

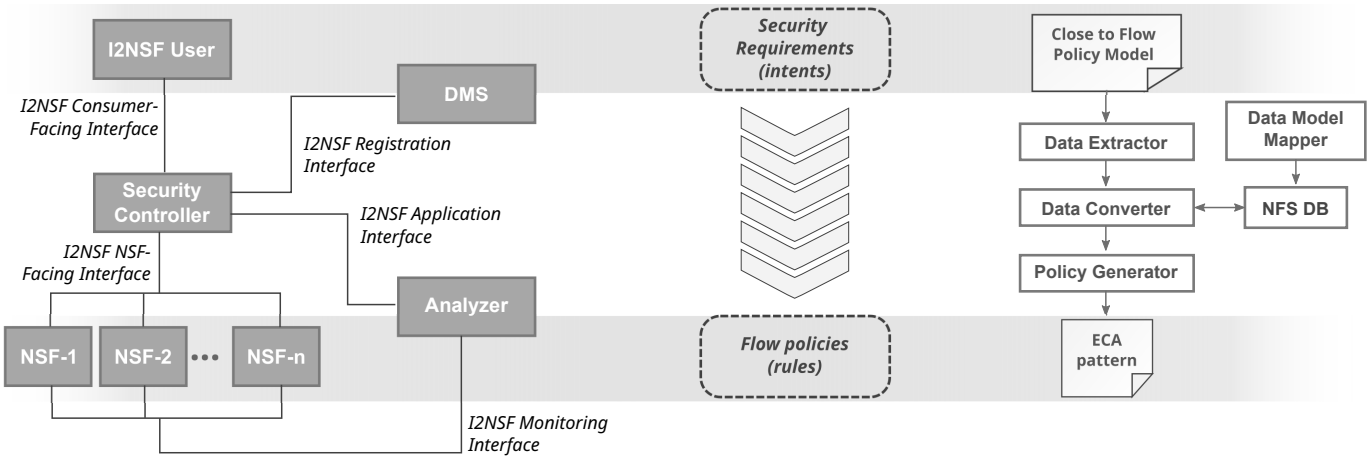
I2NSF policies and configuration rules are expressed with the Event-Condition-Action (ECA) pattern:

- An *Event* triggers the evaluation of the policy/rule. It may be a security alert, a hardware alarm, or a time period.
- A *Condition* is evaluated when the event occurs to check some context, for instance packet headers and content, hardware devices, users or groups.

[1] See https://datatracker.ietf.org/wg/i2nsf/documents/ (accessed: 18th August 2022).

[2] Yet Another Next Generation (YANG) is a data modeling language conceived for information to be transferred over network management protocols like NETCONF and RESTCONF.

[3] eXtensible Markup Language (XML), a language and file format to encode data in both a human- and machine-readable way.



(a) I2NSF architecture (b) Translation of intents to rules (c) Policy translation workflow

Fig. 1. Overview of the I2NSF framework.

- An *Action* is executed if the condition is satisfied. Actions include filtering on ingress packets (pass, drop, mirror, rate-limit), transformation on egress packets (encapsulation, forwarding, signaling), and logging.

In the following, we give a brief overview of each interface, with indication of the main elements in the corresponding data models.

1) Registration interface: The Registration interface is used to feed the SC with available NSF types. It includes two messages. The *Registration* message is sent by the DMS to push the description of new or updated NSFs to the SC, including their capabilities. The *Query* message is used by the SC to look for an NSF in the DMS that has specific capabilities, which are not available for any of the previously registered NSFs. There are two sets of capabilities in the data model, related to security and performance. Security capabilities include the different types of events, conditions and actions supported by the NSF. The list of possible events include both alerts generated by cyber-defense appliances and hardware alarms. There are both per-packet and per-session conditions, covering packet headers, message signatures, content (e.g., Uniform Resource Locators (URLs), call identifiers in VoIP, user agents), target types (computer, tablet, smartphone, etc.), user or group identifiers and geographical locations. Supported actions may include ingress filtering, egress transformation, and logging. Performance capabilities indicate the average and maximum load sustainable by the NSF and are expressed in terms of both computing power and packet processing.

2) Consumer-Facing interface (CFI): This interface should allow I2NSF users to specify security policies in the most technologically-neutral manner, although the current data model only provides limited abstraction with respect to the original design [6]. A policy is a set of ECA rules that apply to a given group of targets, either users or devices. Events refer to specific classes of sources (e.g., anti-Denial-of-Service (DoS), IPS, URL filtering, antivirus, VoIP) and time intervals, whereas Conditions consist of typical parameters for such sources and generic content inspection (payload, URL, context). Finally, Actions distinguish between primary enforcement operations and secondary logging operations. Figure 2(a) shows the structure and main fields of the data model for the CFI.

3) NSF-Facing interface (NFI): This interface defines a vendor-neutral syntax to express security policy rules for NSFs. The overall structure and information of its data model, shown in Fig. 2(b), is rather similar to the CFI. In particular, both Event, Condition, and Action data models can be directly mapped to the corresponding definitions in the CFI.

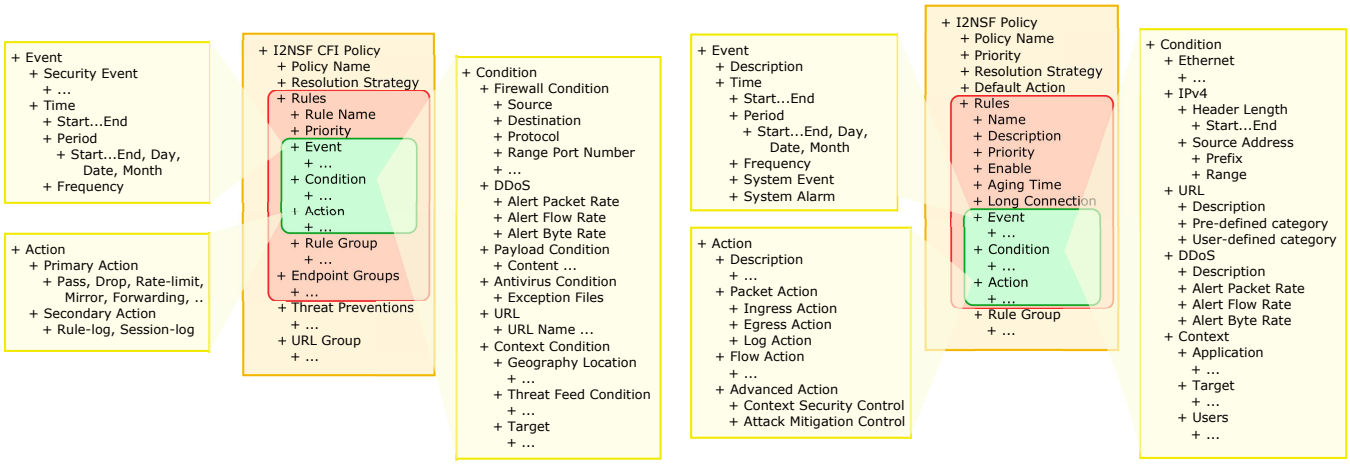
4) Monitoring interface: The range of collected information extends to system/NSF alarm and alert *events*; access, activity and utilization log *records*; system, interface, and NSF *counters*. As regard to the communication model, both subscription/notification and query mechanisms are allowed; the notification can happen immediately or be delayed, so to reduce the number and frequency of (aggregated) messages.

5) Application interface: This interface is used to automatically trigger the SC based on the feedback collected by NSFs. There are two kinds of messages expected on this interface. The first one is for *Policy Reconfiguration* and bears either new or augmented ECA rules generated by the Analyzer; this message is intended to automatically implement mitigation or response actions. The second message, denoted as *Feedback Information*, is generated in case of anomalies that cannot be solved by a policy, such as resource over-usage, malfunctioning, failures.

C. Policy translation

This workflow, shown in Fig. 1(c), is implemented by the SC.

The Data Extractor parses XML messages (high-level policies) and implements a state machine based on Deterministic Finite Automaton (DFA) to move across the different containers, labels, and data. The Data Converter translates the extracted data to the configuration of NSFs, according to the mapping generated by the Data Model and stored in the NSF database. The database contains two kinds of information: endpoint information and NSF capability information. The



(a) Consumer-Facing interface (b) NSF-Facing interface

Fig. 2. Comparison between the policy data model in the CFI and NFI, with detail of the Event, Condition and Action fields.

former is a translation of abstract objects (e.g., "John's computer", "Illegal site") into concrete values (e.g., IP addresses or URLs), as supplied by the I2NSF user. The latter is provided by the DMS, and used to select the appropriate NSF. Finally, the Policy Generator emits the XML with the rules for the NFI.

IV. Open Control and Command

OpenC2 was born to make coordination in cyber-relevant time between decoupled blocks that perform cyber-defense functions. The purpose is to support technology diversity, which undeniably introduces an extra layer of security, but with standardized function-centric interfaces that make security automation and orchestration feasible and less complex to achieve. Differently from I2NSF, OpenC2 is more focused on Machine-to-Machine (M2M) communication for cyber-defense components and only implements the Acting part of the Integrated Adaptive Cyber Defense (IACD) framework, hence leaving sensing, analytics, and decision-making out of scope.

A. OpenC2 framework

The OpenC2 communication stack is layered as shown in Fig. 3, which shows the scope, purpose, main protocols, and name for each layer. At the top of the stack, the Content layer includes both common and function-specific language elements, as defined by the *language specification* [13] and *actuator profiles* [14], respectively. The Message layer defines how OpenC2 content is encoded (as JSON[4] objects) and encapsulated into different transfer protocols, like HTTP, CoAP, AMQP. The *transfer specification* (e.g., OpenC2 over HTTPS [15]) maps metadata and content elements into specific protocol elements (e.g., HTTP header and body, respectively). Finally, Secure Transport conveys data in a secure way over common Internet protocols (e.g., TLS/TCP).

[4]JavaScript Object Notation (JSON) is another human- and machine readable text format for storing and transporting data.

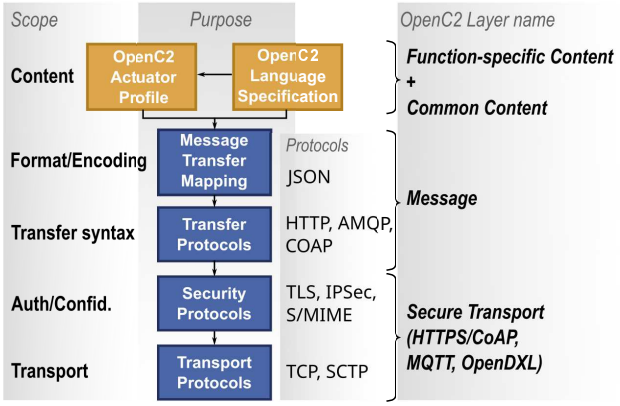


Fig. 3. OpenC2 layering model. Orange boxes falls under the scope of OpenC2, blue boxes are existing protocols.

As the name suggests, OpenC2 is designed to send commands to remote cyber-defense appliances. The communication happens according to a client/server model, where a *Producer* sends *commands* to a *Consumer*, which executes them; the same entity can play both roles. In response to commands, Consumers may generate a *response*, which carries the result of executing the command.

An OpenC2 message is made of a header and a payload (also indicated as "message body," see Listing 1). The header is composed of metadata, including content type, message type, status, request identifier, creation date, origin and destination; it is assembled at the Message layer as defined by the transfer specification. The payload bears the OpenC2 content within an "openc2" object, and it follows the syntax defined in the OpenC2 language. A relevant example of transfer protocol is the Hyper-Text Transfer Protocol (HTTP): OpenC2 commands are sent with the POST method to a well known path (`/.well-known/openc2`), and the header and payload are carried in the body of the HTTP message.

```
POST /.well-known/openc2 HTTP/1.1
Content-type: application/openc2+json;version=1.0
Date: Wed, 19 Dec 2018 22:15:00 GMT
X-Request-ID: d1ac0489-ed51-4345-9175-f3078f30afe5

{
 "headers": {
  "request_id": "d1ac0489-ed51-4345-9175-f3078f30afe5",
  "created": 1545257700000,
  "from": "oc2producer.company.net",
  "to": [
   "oc2consumer.company.net" ]
  },
 "body": {
  "openc2": {
        "request": {
         "action": "deny",
         "target": {
          "ipv4_connection": {
                "protocol": "tcp",
           "src_port": 21
          }
    },
         "args": {
          "slpf": {
           "drop_process": "none",
                "direction": "ingress"
          }
    },
         "actuator": {
          "slpf": {}
    }
       }
  }
 }
}
```

```
HTTP/1.1 200 OK
Date: Wed, 19 Dec 2018 22:15:10 GMT
Content-type: application/openc2+json;version=1.0
X-Request-ID: d1ac0489-ed51-4345-9175-f3078f30afe5

{
 "headers": {
  "request_id": "d1ac0489-ed51-4345-9175-f3078f30afe5",
  "created": 1545257710000,
  "from": "oc2consumer.company.net",
  "to": [
   "oc2producer.company.net" ]
  },
 "body": {
  "openc2": {
        "response": {
    "status": 200,
         "status_text": "Rule correctly inserted"
   }
  }
 }
}
```

Listing 1: Example of OpenC2 command and response for denying inbound FTP connections.

B. OpenC2 language

The language specification [13] defines the common syntax for OpenC2 payloads, in a transfer-agnostic way. In this respect, there are two distinct payload structures: *Command* and *Response*.

A Command (top of Listing 1) contains the following elements:

- **Action**: the instruction, task or activity to be performed (e.g., start, stop, locate, set, update, create).
- **Target**: the object of the action, which is the logical entity affected by the execution of the instruction (e.g., a file, a domain name, an email, a feature, a network connection, a device).
- **Arguments**: additional information on how the command is performed (e.g., time interval, duration, periodicity).
- **Actuator**: the entity that executes the action (e.g., firewall).

Both Targets and Actuators can be specified with different levels of granularity, namely they may identify either a specific object, or a list or a group of objects.

The syntax elements of OpenC2 follows typical language patterns, with a subject (Actuator), a verb (Action), an object (Target), and complements (Arguments). Each element is defined in terms of primitive types (e.g., binary, integer, string) and derived structures (e.g., array, map, enumerated). Structures are recursively used and coupled with semantic constraints to derive typical data objects, as IP/MAC addresses, emails, domain names, dates and times, digests, etc.

A Response (bottom of Listing 1) contains the following elements:

- **Status**: An integer that represents the exit status of the command execution.
- **Status text**: A human-readable description of the status.
- **Result**: a list of key/value pairs produced by the execution of the command.

Even though the OpenC2 language is transfer-agnostic, status code values substantially follow those already defined for HTTP.

Serialization is necessary for interoperability with transfer protocols. The language specification demands the support for JSON serialization and defines the corresponding data type mapping.

C. OpenC2 profiles

The list of Actions provided by the language specification accounts for a broad range of discovery, control, and manipulation operations. However, each Action can only be meaningfully applied to a subset of Targets and implemented by specific Actuators. Moreover, maintaining a common syntax for the great heterogeneity of security functions and their implementations is largely impracticable. Actuator profiles define semantic constraints and language extensions for specific cyber-defense functions; currently, there are several working documents for different appliances, but only the profile for stateless packet filtering has been formally published [14].

An OpenC2 profile defines which combinations of Actions and Targets are relevant for a specific Actuator, including the corresponding Arguments, in the form of a *command matrix*. A profile may also include extensions to the basic syntax, which account for specific features and characteristics of the Actuator. For instance, the Stateless Packet Filtering (SPF) profile includes a “rule number” target which is typically used to update/delete filtering rules, and specific arguments to apply dropping strategies and distinguish between incoming and outgoing packets.

V. Comparison

Even if both I2NSF and OpenC2 are conceived for remote configuration of cyber-defense appliances, they follow rather different designs and approaches, as explained in the following and briefly summarized in Table I.

TABLE I
COMPARISON BETWEEN I2NSF AND OPENC2.

	I2NSF	OpenC2
Scope	Full IACD (claimed)	Remote control (executing course-of-action)
Primary domain	Networking (NFV)	Cyber-defense (IoT)
Interfaces	User-to-Machine (CFI) Machine-to-Machine (NFI, Monitoring, Application, Registration)	Machine-to-Machine only
Encoding	XML	JSON (+ any)
Transport	RESTCONF/NETCONF	Any (HTTP, CoAP, AMQP, ...)
Paradigm	Declarative (intents/high-level policies)	Imperative (verb + objects)
Configuration scope	Load behavioral ECA rules	Broad list of actions (verbs)
Capability	Dynamic (Registration)	Static (Actuator Profiles)
Built-in security	SSH (NETCONF), TLS (RESTCONF) NETCONF Access Control Model (NACM)	TLS, IPSec, S/MIME Coarse-grained access control (HTTP)
Community	I2NSF Working Group (Closed) Several open-source repositories (mostly outdated)	OpenC2 Technical Committee (Open). One software repository

A. Scope

The I2NSF framework aims at covering the whole IACD scope, from sensing to analytics, decision-making, and response; instead, OpenC2 explicitly narrows the scope down to remote control only. However, the current definition of I2NSF is largely network-biased, with fine-grained conditions for packet headers but really bare models for control of antivirus, Distributed Denial-of-Service (DDoS), IPS, VoIP/Voice-over-LTE (VoLTE). This approach makes it more suitable to address simple network threats (mostly limited to volumetric DoS) rather than complex and multi-vector attacks, which exploit encrypted channels, slow-DoS, malicious payloads, software vulnerabilities, and malware.

The OpenC2 approach is more general and easier to extend; however, current profiles restrict its usability to almost the same scenarios as I2NSF.

B. Control models

The control model of I2NSF is largely limited to ECA rules that define the local behavior in response to specific events and conditions, and does not include configuration aspects as algorithms, parameters, Indicator of Compromise (IoC), signatures. Furthermore, ECA rules are largely triggered by hardware/software alarms (e.g., CPU, memory, or disk failures; configuration change) and rather generic security alerts (e.g., list of URLs, packet/flow rate), which are typically used for network management, but do not reflect well the heterogeneity and granularity of events generated by common cyber-defense appliances in terms of IoC. The only practical examples concern filtering of individual packets or HTTP/VoIP connections based on network address/port numbers and URLs/user identifiers, respectively, which can be both traced back to packet filtering.

OpenC2 provides a broader range of possible control actions than plain ECA rules, including scanning files, loading configuration files, locating objects, starting/stopping tasks, setting/updating configuration values, creating/deleting entities, and much more. Even if the scope is deliberately narrowed down to response only, the expressiveness of OpenC2 is indeed suitable for controlling monitoring and detection processes as well, provided that some other kind of mechanism is available to collect and deliver data.

C. Data models

There is a substantial difference in data modeling between I2NSF and OpenC2. Indeed, I2NSF mostly revolves around data models that define *what* the remote appliance is expected to do, and assumes a common method to transfer such configuration. On the contrary, OpenC2 defines *how* the remote appliance is configured but basically disregards the actual form and content of the configuration.

In practical terms, this means that OpenC2 can be virtually applied to any kind of cyber-defense appliances, because it provides generic actions for loading configuration files, setting variables, and starting/stopping tasks. The definition of specific Actuator profiles may be necessary for those applications that either are configured by a command-line interface (CLI) (and cannot load or reload data from a configuration file) or have common usage patterns (e.g., setting rules for a packet filter).

Usage of I2NSF is instead more cumbersome, because of the need to define specific data models for each appliance, and this is poorly scalable due to their heterogeneity in terms of capability and features, often introduced in an incremental manner. The definition of data models for each NSF is also problematic when the same function is provided by different appliances, for instance packet filtering that can be available both in a firewall or IPS. Additionally, extensions from vendors are expected to leverage unique features of their products, but this jeopardizes interoperability and diversity in real systems.

VI. APPLICATION SCENARIOS AND RESEARCH DIRECTIONS

Interfaces to security functions are key to implement adaptive Security Orchestration and Automated Response (SOAR) frameworks, where time-consuming and repetitive tasks are

largely automated by plain behavioral rules or more advanced forms of AI [10]. Control interfaces allow to continuously and automatically tailor the behavior of security functions to fast-changing computing environments (where applications and devices are added, removed, or changed) and the ever-evolving threat landscape.

Fig. 4 shows how I2NSF and OpenC2 can be used in a SOAR framework. It includes the whole workflow to translate high-level policies into low-level configuration rules for remote security functions. This automation is provided by a SC, which continuously update them at run-time according to evolving threats (e.g., retrieved by Cyber-Threat Intelligence (CTI) or vulnerability databases) and security alerts coming from the battery of detectors and analytics engines provided by a Security Information and Event Management (SIEM). A Dashboard is used to control the behavior of the SC, to supervise the operation of the SIEM, and to investigate anomalies.[5]

As already discussed, the scope of OpenC2 is limited to the control interface of security functions, whereas I2NSF fully covers the whole workflow, even if we argued that it is largely network-biased and would not fit the more general scenarios for SOAR. We explicitly indicated the presence of a Local Control Agent that manage security functions; it plays the Producer role in OpenC2 and implements the NFI in I2NSF. We assume that the description of security capabilities (i.e., I2NSF Registration interface) is provided through the management Dashboard, since there is no implementation of a DMS available – indeed, this function is currently mapped to the Element Management (EM) of the Management and Orchestration (MANO) architecture by current I2NSF drafts, which means it should be provided by each VNF vendor.

A. Limitations and complementary aspects

Although I2NSF provides all the functional elements to implement the whole workflow, it currently lacks concrete specifications and technologies to fully implement it. This is the case, for instance, of the CFI, which is currently mapped to the NFI in a rather straightforward way and does not provide the high-level abstraction that was originally expected. The Analyzer is another relevant example, which implementation is not trivial and therefore the current design is mostly limited to forward unchanged information from the monitoring interface. We already noted that the ECA model is more suitable to network management rather then cyber-defense operations, since an I2NSF event is specifically intended to trigger an I2NSF policy rule and not to be processed as an IoC by correlation and detection engines. Now we also argue that the I2NSF Monitoring Interface is not suitable for continuous streaming and indexing of security events and alerts, where the Elastic Stack and similar frameworks already provide better and widely adopted solutions.

OpenC2 provides a better grammar and syntax to control remote agents, beyond the mere "push" operation that is intrinsic in I2NSF, hence it could represent a better transport protocol for I2NSF data models. OpenC2 allows smoother interaction with security functions that expose an Application Programming Interface (API): while I2NSF encodes control actions in ECA rules and delegates each security function to autonomously react to external stimuli, OpenC2 keeps this logic in the SC of the SOAR platform. This approach enables more coordinated orchestration of security functions, by using system-wide context to take decisions, hence following the general principle behind software-defined networking. However, concrete impact of OpenC2 is currently hindered by the lack of uniform profiles for security functions, starting from simpler monitoring and enforcement agents to more complex detection appliances (antivirus, IPS/IDS, etc.). This ultimately jeopardizes most of the automation expected in the SC for decision and response.

Both I2NSF and OpenC2 already provide mechanisms to retrieve the capabilities of cyber-defense functions, but they currently lack the ability to discover such functions deployed within an infrastructure or digital service chain.[6] With agile cloud and NFV services, discovery becomes an essential feature to know the placement and capabilities of security functions every time the system composition or topology changes.

As of the current status, OpenC2 looks more mature than I2NSF; the latter has still many draft documents, which are often contradictory in the terminology and functions and do not seem to get contributions from a large community. Unfortunately, there is no concrete example or use case of applications of these standards to real systems.

B. Future research directions and challenges

In our opinion, future research should mostly focus on addressing OpenC2 limitations. Although I2NSF provides a more comprehensive framework, it suffers from a pronounced network-biased approach, which mostly limits its scope and significance to telco operation. Extensions to more general domains would be worth, because both commercial and open-source SOAR tools are already emerging with a broader scope, targeting better coordination between the different cyber-security teams that work in Security Operation Centers, Incident Response, and Threat Intelligence. [7] More automation is also expected beyond the execution of static response playbooks, in the direction of smart systems that can adapt monitoring, detection, and response processes to complex multi-domain environments and service chains [2], [10].

Indeed, a common interface to security functions is still the missing tile, which still requires to address two main challenges. First, automatic discovery of security functions, in addition to self-description of available capabilities already implemented by I2NSF and OpenC2. Extensions are also necessary for precise identification of security functions, especially when multiple instances are present for load balancing (even on the same node) or better coverage of the service topology.

[5]An example of such framework is provided by the GUARD project [10], https://guard-project.eu/ (accessed: 18th August 2022).

[6]Indeed, the I2NSF Registration interface only discovers the *definitions* of NSFs, not their deployed *instances*.

[7]Extend the power of SOAR with Threat Intelligence Management, video from Palo Alto Networks, March 2020. Available: https://www.paloaltonetworks.com/resources/videos/cortex-xsoar-threat-intelligence.

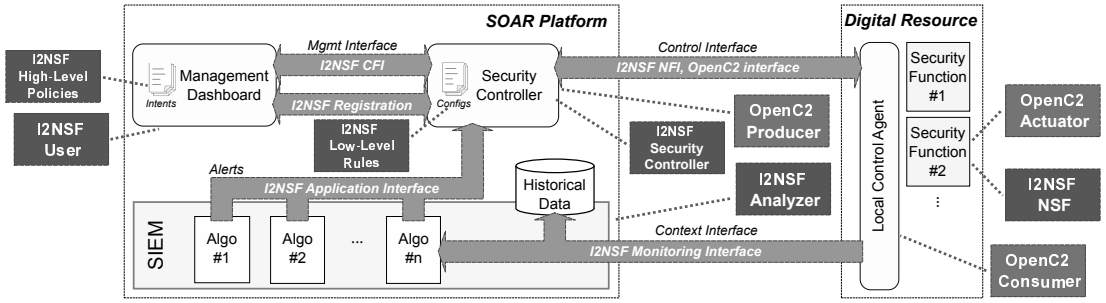


Fig. 4. Mapping of I2NSF and OpenC2 functional elements and interfaces to a typical architecture for detection and mitigation of cyber-attacks.

Second, the definition of common yet extensible abstraction models for addressing the large heterogeneity of security functions, from plain agents to fully-featured appliances. The scope should include both security controls and data, in order to support the design of response logic that is agnostic of the underlying implementation of security functions.

The experience gained with I2NSF may be brought to OpenC2, at least for what concerns network modeling and local behavioral rules. For instance, network traffic may be steered across multiple VNFs for deep packet inspection (e.g., URL or other application-specific properties), traffic classification (e.g., based on protocol headers), and filtering. This possibility is already partially present in I2NSF, but should be considered in OpenC2 as well. However, such integration is not straightforward with current technologies, because existing security functions does not understand ECA rules. Indeed, the open challenge remains the modeling of very heterogeneous security functions, in terms of capabilities and implementations, while avoiding the proliferation of profiles. This requires a research effort in creating a common taxonomy that identifies a limited number of security functions, defines common controls and configuration parameters for each class, and maps them to existing tools.

VII. Conclusion

Our analysis has pointed out the different posture in the design of I2NSF and OpenC2, which largely reflects the main interests and objectives of the corresponding communities. In our opinion, the imperative approach of OpenC2 is winning, because it perfectly fits existing gaps in SOAR frameworks; instead, I2NSF suffers from the pronounced network bias. The narrower scope of OpenC2 has quickly led to a few stable documents, whereas I2NSF is still striving to conclude and give coherence to many working drafts.

Our recent experience in the design of flexible and adaptable frameworks for the detection of cyber-attacks has demonstrated that standard interfaces are really necessary to effectively cope with a large and heterogeneous set of monitoring and enforcement functions, without getting lost in the implementation of an overwhelming number of plugins and adaptation mechanisms. Future work will include contributions to OpenC2 in terms of additional profiles and experimentation of its usage in a concrete use case.

References

[1] M. Repetto, A. Carrega, and R. Rapuzzi, "An architecture to manage security operations for digital service chains," *Future Generation Computer Systems*, vol. 115, pp. 251–266, February 2021.

[2] L. Cui, F. P. Tso, and W. Jia, "Federated service chaining: Architecture and challenges," *IEEE Commun. Mag.*, no. 3, pp. 47–53, March 2020.

[3] V. Mavroeidis and J. Brule, "A nonproprietary language for the command and control of cyber defenses – OpenC2," *Computers & Security*, vol. 97, no. 101999, October 2020.

[4] M. Repetto, G. Bruno, J. Yusupov, G. Lamanna, B. Ertl, and A. Carrega, "Automating mitigation of amplification attacks in NFV services," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 2382–2396, September 2022.

[5] D. Montero, M. Yannuzzi, A. L. Shaw, L. Jacquin, A. Pastor, R. Serral-Gracià, A. Lioy, F. Risso, C. Basile, R. Sassu, M. Nemirovsky, F. Ciaccia, M. Georgiades, S. Charalambides, J. Kuusijärvi, and F. Bosco, "Virtualized security at the network edge: a user-centric approach," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 176–186, April 2015.

[6] S. Hyun, J. Kim, H. Kim, J. Jeong, S. Hares, L. Dunbar, and A. Farrel, "Interface to network security functions for cloud-based security services," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 171–178, January 2018.

[7] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. S. Netto, A. N. Toosi, M. A. Rodriguez, I. M. Llorente, S. De Capitani Di Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M. D. De Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A. Y. Zomaya, and H. Shen, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Computing Surveys*, vol. 1, no. 5, p. 105:1–105:38, September 2019.

[8] L. Bondan, T. Wauter, B. Volckaert, F. De Turck, and L. Zambenedetti Granville, "NFV anomaly detection: Case study through a security module," *IEEE Commun. Standards Mag.*, vol. 60, no. 2, pp. 18–24, February 2022.

[9] S. V. Morais, A. Oliveira de Sá, L. F. Rust, and C. M. Farias, "Malicious traffic description: Toward a data model for mitigating security threats to home IoT," *IEEE Commun. Standards Mag.*, vol. 5, no. 3, pp. 48–55, September 2021.

[10] A. Carrega, G. Grieco, D. Striccoli, M. Paoutsakis, T. Lima, J. I. Carretero, and M. Repetto, *Cybersecurity of Digital Service Chains: Challenges, Methodologies and Tools*, ser. Lecture Notes in Computer Science. Springer Nature, April 2022, vol. 13300, ch. A Reference Architecture for Management of Security Operations in Digital Service Chains, pp. 1–31.

[11] S. Hares, D. Lopez, M. Zarny, C. Jacquenet, R. Kumar, and J. Jeong, "Interface to network security functions (I2NSF): Problem statement and use cases," IETF RFC 8192, July 2017. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc8192.txt.pdf (Accessed 2022-08-18).

[12] D. Lopez, E. Lopez, L. Dunbar, J. Strassner, and R. Kumar, "Framework for interface to network security functions," IETF RFC 8329, February 2018. [Online]. Available: https://tools.ietf.org/pdf/rfc8329 (Accessed 2022-08-18).

[13] "Open command and control (OpenC2)," OASIS standard, November 2019, language Specification Version 1.0, Committee Specification 02.

[14] "Open command and control (OpenC2) profile for stateless packet filtering," July 2019, version 1.0, Committee Specification 01.

[15] "Specification for transfer of OpenC2 messages via https," July 2019, version 1.0, Committee Specification 01.

ACKNOWLEDGMENT

This work has received funding from the European Commission, Grant Numbers: 786922 (ASTRID) and 833456 (GUARD).

BIOGRAPHY

Matteo Repetto received his Ph.D. degree in Electronics and Computer Science from the University of Genoa in 2004. He is currently Senior Researcher at the Institute for Applied Mathematics and Information Technologies of the National Research Council of Italy. Recently, he served Scientific and Technical Coordinator of the EU projects ASTRID (grant no. 786922) and GUARD (grant no. 833456). His research interests include mobility in data networks, virtualization and cloud computing, network functions virtualization and service functions chaining, network and service security.