Chapter 8

A NEW SCAP INFORMATION MODEL AND DATA MODEL FOR CONTENT AUTHORS

Joshua Lubell

Abstract The Security Content Automation Protocol (SCAP) data model for source data stream collections standardizes the packaging of security content into self-contained bundles for easy deployment. However, no single data model can satisfy all requirements. The source data stream collection data model does not adequately meet the needs of SCAP content authors, and its implementation-specific syntax lacks the ability to express packaging subtleties critical to software developers and content authors. This chapter defines a new implementation-neutral information model that is easier to understand and does a better job at expressing relationships between objects comprising a source data stream collection. A new authoring data model for facilitating the implementation of SCAP content development software applications is derived from the information model. Also described is an application implementing the authoring data model that enables SCAP content developers to create source data stream collections using a friendly and intuitive syntax, which is then transformed into SCAP-standard-conforming content.

Keywords: Security Content Automation Protocol, information model, data model

1. Introduction

The Security Content Automation Protocol (SCAP – pronounced *ess-cap*) is an ecosystem of interoperable Extensible Markup Language (XML) [31] vocabularies, reference data repositories and software tools [24]. System administrators – and increasingly operators of manufacturing facilities – use SCAP to secure servers, workstations, networks and other deployed hardware and software. A central part of the SCAP ecosystem is the source data stream collection format, an XML-expressed data model specified in NIST Special Publication (SP) 800-126 (Technical Specification for the Security Content Automation Proto-

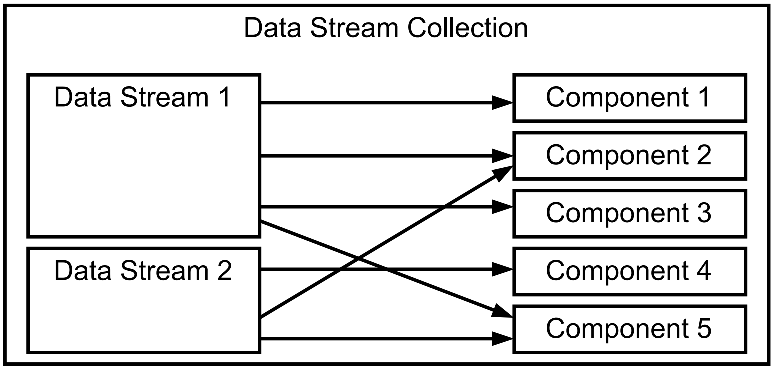

Figure 1. SCAP data stream collection.

col) [28]. This data model is instrumental for enabling the lossless exchange of security content between SCAP-conforming software products. However, no model can single-handedly satisfy the needs of SCAP software developers, content authors and users. This chapter provides an overview of the NIST SP 800-126 source data stream collection data model – highlighting where it succeeds and where it falls short – and then defines supplemental models to address unmet requirements.

The NIST SP 800-126 data model for source data stream collections defines how to package (into a self-contained entity) the collective input required for an SCAP software tool to perform one or more use cases. SCAP use cases include cyber security functions such as configuration checking and vulnerability detection. Self-containment is advantageous because it facilitates SCAP deployment where network connectivity and filesystem access are restricted, as is often the case for industrial control systems and Industrial Internet of Things (IIoT) environments. Self-containment also promotes portability – a single SCAP source data stream collection is easier to distribute reliably to partners, customers and other third parties than an interdependent set of resources. Self-containment also supports digital signing of a source data stream collection as a whole in order to ensure integrity and trustworthiness.

Figure 1 shows the high-level logical relationships within a sample SCAP source data stream collection. The example has two data streams and five components. Each component contains XML data conforming to an XML language that is part of the SCAP ecosystem. Each data stream corresponds to a specific SCAP use case. The arrows pointing from data streams to components are component references. Multiple data streams can reference the same component. For example, both the data streams reference Components 2 and 5.

An SCAP source data stream collection bundles components together such that the components themselves are unmodified from their original states. The packaging operation is thus reversible, allowing for the extraction of SCAP content from a collection and the repackaging of content into a new collection

Table 1. SCAP data model GUID format convention [28].

Object	**Identifier Format Convention**
Data Stream Collection	scap_*reverseDNS*_collection_*name*
Data Stream	scap_*reverseDNS*_datastream_*name*
Component Reference	scap_*reverseDNS*_cref_*name*
Component	scap_*reverseDNS*_comp_*name*

while simultaneously preserving the original content. Reversibility is a desirable property because it promotes interoperable data streams. For example, suppose an SCAP content developer extracts all the components from a source data stream collection, including a security checklist component that conforms to the Extensible Configuration Checklist Description Format (XCCDF) specification [29]. Suppose the user then employs an XCCDF-compliant software tool to select a subset of the checklist rules, assign parameters to the rules and save the resulting XCCDF profile as a separate tailoring component. A tailoring component enables a named profile to be defined separately from the original checklist (without modifying the XCCDF checklist), but it is still explicitly traceable to the original. Next, following best practices for reusing third-party-developed SCAP content [3], the user repackages the extracted components plus the newly-created tailoring component into a new SCAP data stream collection. The reversibility property ensures that none of the components extracted from the old collection and deployed in the new collection are altered.

SCAP encourages content developers to provide globally-unique identifiers (GUIDs) for data stream collections, data streams, components and component references. To this end, the data model requires the identifier format conventions shown in Table 1. An identifier must be an underscore-delimited string beginning with *scap*, followed by a reverse domain name system (DNS) style substring associated with the content author, followed by a substring denoting the object type being identified (*collection*, *datastream*, *cref* or *comp*), and ending with an XML NCName. An NCName [32] is any allowable XML name that does not contain the ":" character. For example, a data stream collection developed by Example Corporation for Ubuntu Linux version 16.04 (also known as Xenial Xerus) could have *scap_com.example_collection_ubuntu-xenial* as its identifier. By promoting GUIDs, the SCAP specification reduces the likelihood of conflicting identifiers in a source data stream collection and that an SCAP content developer would create identifiers that conflict with identifiers created by other developers from the same organization.

The NIST SP 800-126 data model is beneficial for use in applications that consume source data streams, such as configuration scanners and vulnerability detection software. Self-containment of data streams reduces the need for network connectivity. Reversibility preserves the integrity of SCAP components. GUID conventions reduce name collisions. The XML representation of the data model provides additional advantages. It enables software developers to leverage a wide variety of low-cost XML parsers, validators and transformation tools, saving them the trouble of having to implement this functionality in their own software products. Additionally, the XML representation supports the validation of SCAP content and the verification of software purporting to be SCAP-conforming as being in compliance with the NIST SP 800-126 requirements.

However, the characteristics of the NIST SP 800-126 data model that are positives for SCAP software developers can be negatives for developers of SCAP content:

- The GUID formatting conventions result in long and repetitive identifiers. Shorter, context-sensitive identifiers – although dangerous from a deployment standpoint – make a source data stream collection easier for humans to author and understand.
- The XML syntax favors implementation over human readability. For example, the NIST SP 800-126 data model uses the XML Catalogs [19] syntax to define mappings from external uniform resource identifier (URI) references from within a component to the corresponding location within the context of a data stream. The mappings are needed to meet the reversibility and self-containment requirements. Although many XML tools implement XML Catalogs, the syntax is not human-friendly.
- The XML syntax, although naturally hierarchical, is limited in its ability to express the subtleties of part-whole relationships in an SCAP data stream collection. These subtleties are critical to software developers and content authors alike for understanding SCAP data stream collections.

A single data model for source data stream collections is not enough. Although the NIST SP 800-126 data model meets the needs of SCAP configuration scanner and vulnerability detection software developers, it falls short in meeting the needs of developers who create and manage SCAP content. Therefore, SCAP needs the following additional models:

- **Information Model:** The information model for source data stream collections prioritizes human readability over software implementation.
- **Authoring Data Model:** The authoring data model is designed to create new content and transform it to an SCAP-conforming source data stream collection.

As stated by Pras and Schoenwaelder in RFC 3444 [22], an information model and data model are fundamentally different. An information model is expressed at a conceptual level in order to make the design as clear as possible to anyone trying to understand the model, regardless of the implementation context. Therefore, an information model omits implementation details.

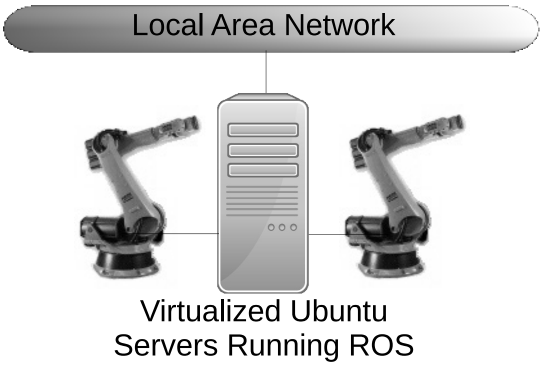

Figure 2. Robots connected to a server running ROS.

In contrast, a data model assumes a specific implementation technology and, therefore, is expressed in an implementation-specific language such as XML. Because a data model is at a lower-level of abstraction than an information model, multiple data models could be derived from a single information model.

This chapter provides an information model for source data stream collections as well an authoring data model. The information model uses the Unified Modeling Language (UML) [16] notation. The authoring data model employs the XML syntax developed using the Darwin Information Typing Architecture (DITA) [20], a standard for authoring, managing, reusing and transforming technical content. Several aspects of the authoring model correspond directly to concepts in the information model described in this chapter, demonstrating the utility of the information model in developing alternative data models. The chapter also describes a software application for creating and transforming an instance of the authoring model into a source data stream collection that conforms to NIST SP 800-126.

The information model, the authoring data model and the authoring and transformation application are all motivated by the growing need for increased SCAP usage in Industrial Internet of Things environments. In this spirit, an example used in the remainder of this chapter is based on a scenario involving a hardware-in-the-loop simulation that is part of a larger industrial control system security testbed [33]. The hardware-in-the-loop simulation involves two robotic arms that interact with a simulated machining process. The simulated manufacturing machines communicate with the robot controllers over a local-area Ethernet network. Each robot is controlled by servers that are deployed as virtual machines within a hypervisor. The controllers run the Robot Operating System (ROS) [7, 26], a software framework widely used in research and increasingly in commercial robotic applications that executes on top of Ubuntu Linux version 16.04. Figure 2 shows the testbed architecture.

The SCAP source data stream collection example used in the context of the testbed scenario incoporates an XCCDF checklist with rules to ensure that AppArmor [2], an Ubuntu Linux kernel enhancement, is installed and configured properly. Ubuntu servers with high security requirements, such as the virtualized servers in Figure 2, commonly use AppArmor. Also the access control method employed by AppArmor works well with ROS [30].

2. Information Model

The source data stream collection information model has the following goals:

- Make compositional relationships more explicit. The UML notation allows for this whereas the XML syntax does not.
- Omit implementation guidance that gets in the way of human understanding, specifically, the GUID conventions. Such guidance is vital for implementations, but it can make models unnecessarily confusing in a pedagogical context.
- Facilitate the development of other models. An information model should pave the way for the development of models that are implementation-focused. The discussion of the authoring model later in this chapter provides examples of how the authoring model elements and attributes correspond to their counterparts in the information model.

Figure 3 shows a UML class diagram representing the source data stream collection information model. A `DataStreamCollection` contains one or more `DataStream` objects and one or more `Component` objects. The `DataStream` and `Component` objects do not exist outside the scope of `DataStreamCollection`, as indicated by the solid diamonds on the links connecting them to the `DataStream Collection`. The `reverseDNS` UML attribute of `DataStreamCollection` has as its value the reverse-DNS string used in SCAP identifiers (see Table 1).

A `Component` contains an object that is a subtype of `XMLDocument`. The `timestamp` UML attribute of a `Component` specifies when the `XMLDocument` was packaged as part of a `DataStreamCollection`. Thus a `Component` is nothing more than a snapshot of `XMLDocument` at a particular point in time. `XMLDocument` is italicized in Figure 3, indicating that it is an abstract class (which cannot be instantiated). `XMLDocument` is a generalization of the five allowable SCAP source data stream component XML document types.

The five subclasses of `XMLDocument` are:

- **CPEDictionary:** This is an XML representation of a platform (hardware, operating system or software application). Each platform has a unique Common Platform Enumeration (CPE) identifier.
- **Benchmark:** This is an XML representation of a security checklist (also called a benchmark), which is valid with respect to the Extensible Configuration Checklist Description Format (XCCDF) specification.
- **OVALDefs:** This is an XML representation of system configuration information, tests and states, which is valid with respect to the Open Vulnerability Assessment Language (OVAL) specification [21]. XCCDF

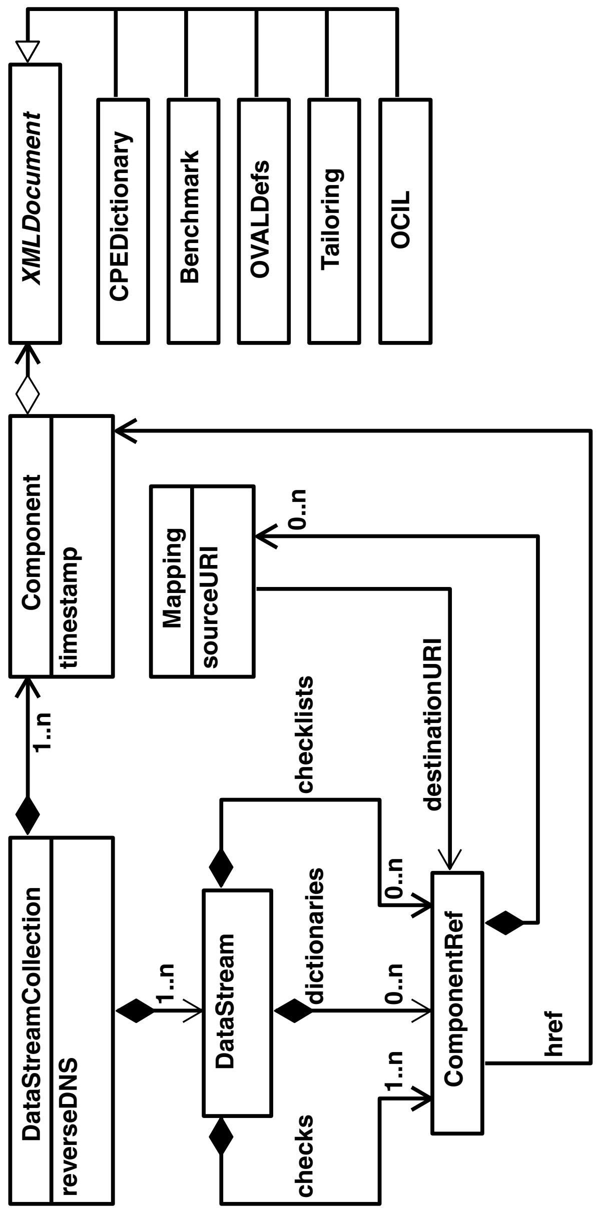

Figure 3. Source data stream collection UML class diagram.

checklist rules use OVAL to determine if the current state of a system satisfies the rule criteria. XCCDF checklist rules and OVAL definitions together typically account for most of the XML data in a source data stream collection.

- **Tailoring:** This is an XML representation of the profiles of a Benchmark, which is valid with respect to the `<Tailoring>` element definition of the XCCDF specification.
- **OCIL:** This is an XML format used by XCCDF rules for checks requiring information collected from a human via a questionnaire. It is valid with respect to the Open Checklist Interactive Language (OCIL) specification. OCIL is used for checking state via human-oriented collection of information that is not feasibly obtained using OVAL-based methods.

The existence of an `XMLDocument` is not limited to its existence in the context of a `Component`, as indicated in Figure 3 by the hollow diamond on the link from `Component` to `XMLDocument`. What this means is that, in addition to being part of the `Component`, the `XMLDocument` can be part of a `Component` in another data stream collection or have a life of its own outside the scope of SCAP. As a consequence, an `XMLDocument` in a source data stream collection may reference another `XMLDocument` in the same collection, but using a URI outside the scope of the source data stream collection. For example, a source data stream collection could incorporate a `Benchmark` and an `OVALDefs` with lives outside the scope of the collection, with the `Benchmark` using an external URI to reference the `OVALDefs`.

The source data stream collection information model handles external URI references in a manner that maintains the SCAP reversibility and self-containment requirements discussed in the introductory section. A `DataStream` contains at least one `ComponentRef` that references a `Component` containing an `OVALDefs` or `OCIL` object, and zero or more `ComponentRef` objects referencing a `Component` containing a `CPEDictionary` object, `Benchmark` object or `Tailoring` object. A `ComponentRef` may contain zero or more `Mapping` objects. A `Mapping` resolves references from within an `XMLDocument` to another `XMLDocument`. The `Mapping` accomplishes this by providing the information needed to translate the URI within the `XMLDocument` referencing the external resource to a URI referencing the `ComponentRef` within the `DataStream` containing the `ComponentRef` to which the `Mapping` belongs. The `sourceURI` UML attribute of the `Mapping` has as its value a URI that matches a referenced URI in the `Component` referenced by the `ComponentRef` that contains the `Mapping`. The `destinationURI` association of the `Mapping` references a `ComponentRef` object.

The UML object diagram in Figure 4 illustrates how the information model in Figure 3 could be used to describe a source data stream collection incorporating the XCCDF checklist introduced above. The XCCDF checklist `xenial-apparmor-xccdf.xml` and its referenced `oval-definitions.xml` are represented as `Benchmark` and `OVALDefs` objects. The `OVALDefs` object is

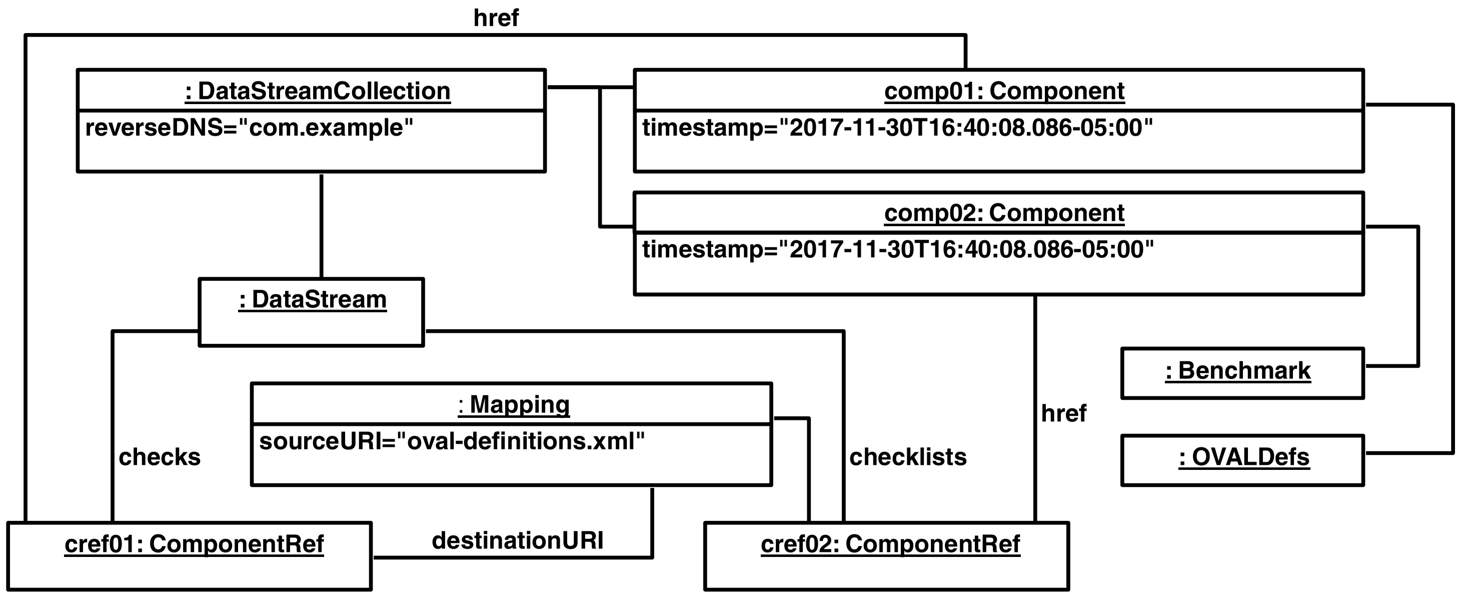

Figure 4. UML object diagram.

contained in `Component` comp01 and the `Benchmark` object is contained in `Component` comp02. `ComponentRef` cref02 contains a `Mapping` object. This mapping object is needed because the XCCDF `<check>` elements of `Benchmark` contain URI references to `oval-definitions.xml`, for example:

```
<check system="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <check-content-ref href="oval-definitions.xml"
                     name="oval:com.ubuntu.xenial:def:100"/>
</check>
```

This `<check>` element specifies that OVAL definition `oval:com.ubuntu.xenial:def:100` should be used to determine compliance with the XCCDF rule containing the `<check>` element, and that the OVAL definition is located at the relative URI `oval-definitions.xml`. The `Mapping` object says that, instead of looking for the OVAL definition in `oval-definitions.xml`, an SCAP-conforming software product processing the `DataStream` object should locate the OVAL definition within `Component` comp01 (referenced by `ComponentRef` cref01).

3. Authoring Data Model and Application

The Darwin Information Typing Architecture (DITA) [20, 23] has two primary building blocks: the *topic* and *map* XML element types. A topic represents a chunk of information. A map represents a collection of topics or other maps. DITA facilitates the reuse of topics and maps, as well as XML elements and fragments within a topic or map. DITA topic and map types are specializable. Specialization, which is the inverse of generalization, helps avoid inconsistencies and enables interoperability [12]. DITA allows for the definition of new specialized element types based on built-in topic and map types. A specialized DITA information type refines the existing base type and, therefore, must be at least as constrained. By adhering to these constraints, specialized DITA types have the advantage that implementations can easily leverage other DITA-conforming implementations [11].

The authoring data model defines a new DITA element type for source data stream collections, which is specialized from the DITA map base type. This new element type is expressed as a DITA document type shell based on DITA's map document type shell. A document type shell defines the elements and attributes that are allowed in a DITA XML document conforming to the specialized element type. The data stream collection document type shell follows the DITA standard's modular architecture for creating shells, ensuring that the shell can be used with any DITA-conformant XML authoring tool.

The DITA map type was chosen as the basis for specialization because an SCAP source data stream collection is inherently map-like. Like a DITA map, a source data stream collection is essentially a structured collection of references to components. Maps can use the DITA `<topicref>` element to reference external (non-DITA) resources, as well as to aggregate groups of nested `<topicref>` elements. Both these uses of `<topicref>` correspond to concepts from the in-

Table 2. Data stream collection DITA document type shell.

Element	Specializes	Content Model

`<DataStreamCollection>`	`<map>`	`@reverseDNS`
		`@scapName`
		`@schematronVersion`
		`<scapComponent>`*
		`<DataStream>`+

`<scapComponent>`	`<keydef>`	`@keys` `@href`

`<DataStream>`	`<topicref>`	`@scapName`
		`@scapVersion`
		`@useCase`
		`<Dictionaries>`?
		`<Checklists>`?
		`<Checks>`

`<Dictionaries>`	`<topicref>`	`<CpeListRef>`+
`<Checklists>`	`<topicref>`	`<BenchmarkRef>`+ `<TailoringRef>`+
`<Checks>`	`<topicref>`	`<OvalRef>`+ `<OcilRef>`+

`<CpeListRef>`
`<BenchmarkRef>`
`<TailoringRef>`
`<OvalRef>`
`<OcilRef>`
	`<topicref>`	`@keyref`
		`<ExternalLinks>`?

`<ExternalLinks>`	`<topicref>`	`<Uri>`+
`<Uri>`	`<topicref>`	`@keyref`

formation model in Figure 3. A `ComponentRef` object references a subclass of `XMLDocument`, which is an external resource. The `DataStreamCollection` composition link pointing to `DataStream` collects `DataStream` objects. The `dictionaries`, `checklists` and `checks` composition links of a `DataStream` collect `ComponentRef` objects. The `ComponentRef` composition link pointing to `Mapping` collects `Mapping` objects. Therefore, the DITA source data stream element type defines new elements specialized from `<topicref>` to represent data streams, component references, collections of component references and mappings from URI references within external resources to the appropriate component references.

Table 2 shows the XML elements and attributes in the source data stream collection document type shell. The left-hand column contains the element names. The middle column presents the DITA map built-in element specialized to define the element in the left-hand column. All the left-hand columns elements are specializations of `<topicref>`, except for `<DataStreamCollection>` (which specializes `<map>`) and `<scapComponent>` (which specializes `<keydef>`, a built-in map element). The right-hand column shows the content model that constrains each left-hand column element. Names preceded by an @-sign are required XML attributes. An asterisk following an element means zero or more occurrences of the element are allowed. A plus sign means one or more occurrences are allowed. A question mark means zero or one occurrences are allowed. For example, `<CpeListRef>` has a required `@keyref` attribute and may optionally contain a single `<ExternalLinks>` element.

The `<scapComponent>` element of the document type shell contains no sub-elements. This is because it has no author-provided content. As mentioned above, a `Component` is no more than an `XMLDocument` with a timestamp added to it. Since the timestamp is system-generated, the authoring and transformation application only needs the referenced XML resources external to the data stream collection DITA map to create NIST SP 800-126 data model component elements.

In order to understand how the authoring and transformation application processes an XML instance in a manner that is valid with respect to the authoring data model, consider the following DITA map, which represents the Xenial AppArmor source data stream collection from Figure 4:

```
<DataStreamCollection reverseDNS="com.example" scapName="apparmor"
                      schematronVersion="1.2">
  <scapComponent keys="xccdf_apparmor"
                 href="xenial-apparmor-xccdf.xml"/>
  <scapComponent keys="oval_apparmor"
                 href="oval-definitions.xml"/>
  <DataStream scapName="xenial_apparmor" scapVersion="1.3"
              useCase="CONFIGURATION">
    <Checklists>
      <BenchmarkRef keyref="xccdf_apparmor">
        <ExternalLinks>
          <Uri keyref="oval_apparmor"/>
        </ExternalLinks>
      </BenchmarkRef>
    </Checklists>
    <Checks>
      <OvalRef keyref="oval_apparmor"/>
    </Checks>
  </DataStream>
</DataStreamCollection>
```

The `@reverseDNS` attribute of the `<DataStreamCollection>` element responds directly to its counterpart in Figure 4. `@scapName` provides the *name* portion needed to construct the NIST SP 800-126 data stream collection identifier according to the GUID conventions in Table 1. `@schematronVersion` specifies the version of the Schematron schema to which the source data stream collection conforms. This information is needed because NIST SP 800-126 requires a source data stream collection to be valid with respect to a set of rules defined using Schematron [10], an XML language for expressing and testing natural language assertions about an XML document type.

The source data stream collection type uses `<scapComponent>` to associate a more succinct key name (`@keys`) with an XML document URI (`@href`). This serves multiple purposes. First, it makes source data stream collection DITA maps easier to maintain. Referencing each URI only once in `<scapComponent>` and referencing the associated name elsewhere in `@keyref` XML attributes add a level of indirection, reducing the number of DITA map revisions needed if an XML document URI changes. Second, using the key name in place of the URI improves readability of the XML. Finally – and most importantly – key names serve as the *name* portion of GUIDs generated by the authoring and transformation application when processing `@keyref` XML attributes.

`<DataStream>` has three attributes: (i) `@scapName` provides the *name* portion used by the authoring and transformation application to construct the data stream GUID; (ii) `@scapVersion` specifies the version of SCAP to which the data stream conforms (1.3 is the most recent SCAP version); and (iii) `@useCase` specifies the SCAP use case.

`<Checklists>`, which corresponds to the *checklists* composition link in Figure 4, contains `<BenchmarkRef>` elements. The authoring and transformation application uses `<BenchmarkRef>` to generate a data stream component that holds the contents of `xenial-apparmor-xccdf.xml` and a component reference. The generated component is simply a wrapper element with an application-generated timestamp value that contains the XCCDF XML. As discussed above, the XCCDF `<check>` elements contain URI references to `oval-definitions.xml`. The generated component reference, where *sds:* and *cat:* are XML namespace prefixes mapping to namespaces defined in [28] and [19], respectively, is as follows:

```
<sds:component-ref
    id="scap_com.example_cref_xccdf_apparmor"
    href="#scap_com.example_comp_xccdf_apparmor">
  <cat:catalog>
    <cat:uri name="oval-definitions.xml"
             uri="#scap_com.example_cref_oval_apparmor"/>
  </cat:catalog>
</sds:component-ref>
```

The `@id` value of the `<sds:component-ref>` element is a GUID generated by the authoring and transformation application using the `@keyref` value of the DITA map's `<BenchmarkRef>`. The `@href` value refers to the `@id` of the `<sds:component>` that contains the XCCDF checklist XML. The transformation generates `<cat:catalog>` from the DITA map's `<ExternalLinks>` element and `<cat:uri>` from the DITA map's `<Uri>` element, which corresponds to the `Mapping` object in Figure 4. The authoring and transformation application assigns the `@href` value of the `<scapComponent>` whose `@keys` attribute value matches `<Uri>`'s `@keyref` value to the `<cat:uri>` `@name` attribute. `<cat:uri>`'s `@uri` attribute is assigned a component reference GUID prefaced by # whose *name* substring is the value of the `<Uri>` element's `@keyref` attribute.

`<Checks>` and `<OvalRef>` are transformed similarly to `<Checklists>` and `<BenchmarkRef>`; however, since OVAL definitions do not reference any external URIs, there is no embedded `<ExternalLinks>` element to transform.

The authoring and transformation application was implemented using the DITA Open Toolkit [5], a specialization-aware, output-producing DITA processor. The DITA standard requires output-producing processors to merge topics referenced in a map as well as resolve key references, eliminating the need for custom transformation code to perform the functions. Specialization-aware DITA processors are required to do all of the above for specialized DITA documents by inheriting processing behavior from base types. Therefore, leveraging the DITA Open Toolkit greatly reduced the coding effort required to build the authoring and transformation application.

The DITA Open Toolkit has a modular architecture with an extensible plug-in mechanism for implementing custom document type shells and output formats. Plug-ins can be run in any XML authoring software environment that uses the DITA Open Toolkit. The authoring and transformation application was implemented as a NIST SP 800-126 conformant output plug-in. The source data stream collection document type shell was also implemented as a plug-in. The authoring and transformation application was successfully deployed in a commercial XML editor product, which was then used to create the Xenial AppArmor DITA map example in this chapter as well as other SCAP source data stream collection examples.

4. Related Efforts and Next Steps

Other recent and ongoing research efforts have fostered the development of systems of related models for achieving automation and integration. Kulvatunyou et al. [13] provide examples of standards for smart manufacturing where alternative models were developed to satisfy different implementation contexts. Smart manufacturing requires all engineering information to be represented digitally and to be completely computer interpretable. Two examples provided by Kulvatunyou and colleagues are ISO 10303-242 [9], a standard for computer-aided design (CAD) geometry and product manufacturing data, and the Open Application Group Integration Specification (OAGIS) [17], a suite of information standards for interfacing manufacturing systems with business functions such as sales and finance. ISO 10303-242 includes a low-level data model for CAD geometry and other CAD-related information, as well as a higher-level business object model that represents additional information needed for manufacturing and product support, such as part assemblies and bills of materials. OAGIS defines an abstract implementation-neutral information model for individual transaction standards called business object documents (BODs). OAGIS also defines multiple data models for implementing business object documents, including an XML-based model and a JavaScript Object Notation (JSON) [4] model.

Health Level 7 (HL7) [8] – an organization that promulgates standards for exchange management and integration of healthcare information – has created a standards architecture with an abstract information model from which implementation-specific data models are derived. The HL7 Reference Information Model (RIM) is broad and minimalist, but it provides an integrated view that facilitates the development of interoperable implementation-specific data models [6]. The Clinical Document Architecture, an HL7 standard derived from the HL7 RIM, combines an XML document type with a specialized RIM-based model to precisely specify clinical information requirements [8].

As part of a study on the challenges of automating security configuration checklists in manufacturing environments, Lubell and Zimmerman [15] developed a simple XCCDF checklist modeled in UML. The UML model uses `AND`, `OR` and `NOT` classes to represent Boolean operations in XCCDF `<check>` elements. In a follow-up effort by Lubell [14], a DITA element type developed for XCCDF rules uses specializations of DITA's built-in `<sectiondiv>` topic element to model Boolean operations. This XCCDF rule element type demonstrates the power and versatility of DITA specialization, and was a precursor to the research presented in this chapter.

The DITA XCCDF rule and SCAP data stream collection element types exemplify the recent trend of using DITA to create and manage intelligent content. Traditional content management solutions focus on information that is consumed mainly by humans via print media, the web or (more recently) mobile devices. Intelligent digital content such as SCAP, however, can be delivered to a broader range of targets for multiple purposes – not just to humans for reading – and, therefore, requires higher-precision data models and increased automation [25]. The increasing prevalence of intelligent content is causing content management to evolve from being mainly editorial in nature to a more engineering-focused pursuit [1].

The research discussed in this chapter leads to two questions that merit future study:

- How effective would the proposed information and authoring data models be in reducing the effort needed to develop and deploy SCAP source data stream collections in the hardware-in-the-loop testbed environment discussed in the introduction?
- Would expanding the scope of the information and authoring models to include low-level objects constituting `Benchmark` and `OVALDefs`, in addition to high-level concepts such as `DataStream` and `Component`, enable an authoring solution that is superior to existing approaches?

To answer the first question, the source data stream collection authoring application could be used to support a NIST-industry collaborative effort to establish best practices for securing industrial control systems in the manufacturing sector [27]. Two cyber security capabilities within the project scope – behavioral anomaly detection and industrial control application whitelisting – are addressable using SCAP. For example, a source data stream could check that AppArmor is installed and properly configured to protect an industrial control system from a software application hijacked by malware that causes the application to behave in an aberrant manner. As another example, a source data stream deployed in an industrial control device could enforce application whitelisting by checking if installed software packages are on an approved whitelist. The information model and authoring-model-based application could be used to assemble a source data stream collection from existing XCCDF rules and OVAL definitions for detecting behavioral anomalies and the presence of unauthorized software. The effort expended could then be compared against the effort required for manual source data stream collection, or against third-party software tools that might be available.

Answering the second question would require more effort than answering the first question and would involve the following modeling and implementation steps:

- Develop information models for XCCDF benchmarks and OVAL definitions. Based on these information models:
  - Create DITA specializations corresponding to XCCDF XML schema elements for representing benchmarks, profiles and rules.
  - Create DITA specializations corresponding to OVAL XML schema elements for representing definitions, criteria, tests and endpoint information.
- Implement an authoring and transformation application that assembles the collection of DITA documents representing the XCCDF and OVAL into a source data stream collection conforming to NIST SP 800-126.

The resulting implementation could then be compared against the current approach used to author and manage content for the SCAP Security Guide (SSG) [18], an open-source project whose output is a set of SCAP source data stream collections for Linux distributions and software applications. Contributors of SCAP Security Guide content use an *ad hoc* collection of tools created by the guide developers for authoring content such as XCCDF checklist rules and OVAL definitions. These tools enable contributors to use a shorthand XML syntax that is transformed into standards-conforming XCCDF and OVAL content, which in turn are transformed into a source data stream collection conforming to NIST SP 800-126. As discussed in [14], although the SCAP Security Guide authoring framework has proven successful in producing extensive and widely-used SCAP content, the framework and tools are complex, difficult for contributors to understand and hard for SCAP Security Guide developers to maintain. They also lack the validation capabilities of DITA document shells and authoring convenience of DITA-specialization-aware XML editing software.

Although the DITA-based approach shows promise [14], more thorough implementation and analysis are needed to determine whether or not the preliminary results are scalable to a larger and more representative corpus of security content.

5. Conclusions

This chapter describes two original research contributions: (i) a UML information model representing SCAP source data stream collections; and (ii) an authoring data model specialized from the DITA map element type and derived from the UML information model. The illustrative example involving the secure configuration of servers that control industrial robots demonstrates that the information model is easier to understand than the XML-based data model described in NIST SP 800-126, and is also better at expressing compositional relationships in a data stream collection. A DITA Open Toolkit plug-in implementation of the authoring data model provides a means for creating new SCAP content in an author-friendly manner and producing output that conforms to NIST SP 800-126. The review of related research reveals parallels with information models and data models developed for manufacturing systems and for healthcare enterprises, as well as with emerging trends in the field of content management.

The Industrial Internet of Things is spurring the need to secure an ever-growing variety of devices, operating systems and software. The diversity requires better tools than those currently available for SCAP content authors. The proposed source data stream collection information model and authoring model constitute a first step toward the development of SCAP authoring and content management solutions that meet the challenges.

This chapter is a contribution of the National Institute of Standards and Technology (NIST). Certain commercial and third-party products and services are identified in this chapter to enhance understanding. Such identification does not imply any recommendation or endorsement by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

Acknowledgement

The author wishes to thank the individuals who provided helpful reviews of earlier drafts of this chapter, especially his NIST colleagues, David Waltermire and Timothy Sprock, for their insights regarding SCAP and information modeling.

References

[1] R. Andersen and T. Batova, The current state of component content management: An integrative literature review, *IEEE Transactions on Professional Communication*, vol. 58(3), pp. 247–270, 2015.

[2] M. Bauer, Paranoid Penguin: AppArmor in Ubuntu 9, *Linux Journal*, issue 185, September 1, 2009.

[3] H. Booth, M. Cook, S. Quinn, D. Waltermire and K. Scarfone, Security Content Automation Protocol (SCAP) Version 1.2 Content Style Guide: Best Practices for Creating and Maintaining SCAP 1.2 Content, NISTIR 8058 (Draft), National Institute of Standards and Technology, Gaithersburg, Maryland, 2015.

[4] T. Bray, The JavaScript Object Notation (JSON) Data Interchange Format, RFC 8259, 2017.

[5] DITA Open Toolkit Project, DITA Open Toolkit (`www.dita-ot.org`), 2018.

[6] T. Eggebraaten, J. Tenner and J. Dubbels, A health-care data model based on the HL7 Reference Information Model, *IBM Systems Journal*, vol. 46(1), pp. 5–18, 2007.

[7] C. Fairchild and T. Harman, *ROS Robotics by Example*, Packt Publishing, Birmingham, United Kingdom, 2016.

[8] Health Level Seven International, About HL7 International, Ann Arbor, Michigan (`www.hl7.org`), 2018.

[9] International Organization for Standardization, Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 242: Application Protocol: Managed Model-Based 3D Engineering, ISO 10303-242:2014, Geneva, Switzerland, 2014.

[10] International Organization for Standardization, Information Technology – Document Schema Definition Languages (DSDL) – Part 3: Rule-Based Validation – Schematron, ISO/IEC 19757-3:2016, Geneva, Switzerland, 2016.

[11] E. Kimber, *DITA for Practitioners, Volume 1, Architecture and Technology*, XML Press, Laguna Hills, California, 2012.

[12] S. Krima and J. Lubell, Flat versus hierarchical information models in PLM standardization frameworks, in *Product Lifecycle Management for Digital Transformation of Industries*, R. Harik, L. Rivest, A. Bernard, B. Eynard and A. Bouras (Eds.), Springer, Cham, Switzerland, pp. 121–133, 2016.

[13] B. Kulvatunyou, N. Ivezic and V. Srinivasan, On architecting and composing engineering information services to enable smart manufacturing, *Journal of Computing and Information Science in Engineering*, vol. 16(3), pp. 031002-1–031002-13, 2016.

[14] J. Lubell, Using DITA to create security configuration checklists: A case study, *Proceedings of Balisage: The Markup Conference*, vol. 19, 2017.

[15] J. Lubell and T. Zimmerman, Challenges to automating security configuration checklists in manufacturing environments, in *Critical Infrastructure Protection XI*, M. Rice and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 225–241, 2017.

[16] Object Management Group, OMG Unified Modeling Language Version 2.5.1, Needham, Massachusetts (`www.omg.org/spec/UML/2.5.1`), 2017.

[17] Open Applications Group, OAGi Integration Specification Release 10.4, Marietta, Georgia (`www.oagi.org`), 2018.

[18] OpenSCAP Project, SCAP Security Guide: Baseline Compliance Content in SCAP Formats (`github.com/OpenSCAP/scap-security-guide`), 2018.

[19] Organization for the Advancement of Structured Information Standards, XML Catalogs v1.1, OASIS Standard, Burlington, Massachusetts (`www.oasis-open.org/standards#xmlcatalogsv1.1`), 2005.

[20] Organization for the Advancement of Structured Information Standards, Darwin Information Typing Architecture (DITA) v1.3, OASIS Standard, Burlington, Massachusetts (`www.oasis-open.org/standards#ditav1.3`), 2016.

[21] OVAL Project, OVAL Documentation (`ovalproject.github.io`), 2017.

[22] A. Pras and J. Schoenwaelder, On the Difference Between Information Models and Data Models, RFC 3444, 2003.

[23] M. Priestley and D. Schell, Specialization in DITA: Technology, process and policy, *Proceedings of the Twentieth Annual International Conference on Computer Documentation*, pp. 164–176, 2002.

[24] S. Radack and R. Kuhn, Managing security: The Security Content Automation Protocol, *IT Professional*, vol. 13(1), pp. 9–11, 2011.

[25] A. Rockley and J. Gollner, An intelligent content strategy for the enterprise, *Bulletin of the American Society for Information Science and Technology*, vol. 37(2), pp. 33–39, 2011.

[26] ROS Industrial Consortium, ROS-Industrial, San Antonio, Texas (`rosindustrial.org`), 2018.

[27] K. Stouffer and J. McCarthy, Capabilities Assessment for Securing Manufacturing Industrial Control Systems, Cybersecurity for Manufacturing, National Cybersecurity Center of Excellence, National Institute of Standards and Technology, Gaithersburg, Maryland, 2017.

[28] D. Waltermire, S. Quinn, H. Booth, K. Scarfone and D. Prisaca, The Technical Specification for the Security Content Automation Protocol (SCAP) Version 1.3, NIST Special Publication 800-126, Revision 3, National Institute of Standards and Technology, Gaithersburg, Maryland, 2018.

[29] D. Waltermire, C. Schmidt, K. Scarfone and N. Ziring, Specification for the Extensible Configuration Checklist Description Format (XCCDF), Version 1.2, NISTIR 7275, Revision 4, National Institute of Standards and Technology, Gaithersburg, Maryland, 2012.

[30] R. White, H. Christensen and M. Quigley, SROS: Securing ROS over the wire, in the graph and through the kernel, presented at the *IEEE-RAS International Conference on Humanoid Robots*, 2016.

[31] World Wide Web Consortium, Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation, Massachusetts Institute of Technology, Cambridge, Massachusetts (`www.w3.org/TR/REC-xml`), November 26, 2008.

[32] World Wide Web Consortium, Namespaces in XML 1.0 (Third Edition), W3C Recommendation, Massachusetts Institute of Technology, Cambridge, Massachusetts (`www.w3.org/TR/xml-names`), December 8, 2009.

[33] T. Zimmerman, Metrics and Key Performance Indicators for Robotic Cybersecurity Performance Analysis, NISTIR 8177, National Institute of Standards and Technology, Gaithersburg, Maryland, 2017.