Security Working Group

An Interoperable Architecture for Security Labeling Rules – Based on CQL and FHIR Clinical Reasoning Module

Mohammad Jafari,
Kathleen Connor, Christopher Shawn

Version 1.1

Sep 22, 2021

Table of Contents

Table of Figures

1 INTRODUCTION

Security labels are privacy/security-related metadata associated with health information which cover a wide-range of concepts such as confidentiality, sensitivity, integrity, obligations, refrains, trust labels, etc. Security labels are an important element in enforcing privacy and security policies, so, most electronic health record (EHR) standards, including Fast Healthcare Interoperability Resources (FHIR), support labeling health information with security labels. Security labels are chosen from standard vocabularies with standard values and agreed-upon semantics. For an overview of various types of security labels and their value-sets see the FHIR Data Segmentation for Privacy (DS4P) Implementation Guide [1].

As consumers of security labels, any application or service that receives a labeled data item should be capable of parsing the security labels, understanding their meaning, and incorporating them in making decisions such as access control decisions, workflow navigation, or user interface rendering.

For providers of health information, a Security Labeling Service (SLS) is used to determine security labels and assign them to data items – or portions of a data item [2].

The SLS receives a data item (e.g., a FHIR resource) or a collection of data items (e.g., a FHIR bundle) and determines the security labels based on a set of Security Labeling Rules.[1] These rules define the criteria that a data item must meet in order to be assigned a particular security label. For example, if in a FHIR `Condition` resource, the `code` attribute is set to Systematized Nomenclature of Medicine Clinical Terms (SNOMED-CT) code `832007` indicating *moderate major depression*, the resource must be assigned the sensitivity label `MH` indicating *mental health information sensitivity*. At the conceptual level, Security Labeling Rules are based on clinical knowledge and privacy policies, and therefore are determined by privacy experts and clinicians. But at the implementation-level these rules must be expressed in a machine-readable form so that the SLS can understand and apply them; therefore a technical model, and a language is needed for expressing these rules.

In the past few years, the conceptual and implementation models of the SLS have been the focus of attention and discussion. Meanwhile, there has not been much discussion on Security Labeling Rules and models for expressing, implementing, maintaining, and exchanging them in a standard manner. A standard way of expressing and communicating Security Labeling Rules paves the way for defining standard libraries of labeling rules based on clinical knowledge and jurisdictional policies by experts; these rules can then be used by different SLS implementations thereby ensuring a consistent and uniform interpretation of the privacy policies and their enforcement.

This paper proposes a model for formulating, managing, and exchanging Security Labeling Rules in a standard and interoperable way by leveraging Health Level Seven (HL7) FHIR concepts and artifacts including Clinical Quality Language (CQL) [4] (and FHIRPath [3]) as well as the resources from the FHIR Clinical Reasoning module [5].

[1] Optionally, additional attributes such as the identity of the recipient of the information or contextual information about the transaction at hand may also be provided to the SLS to make labeling decisions.

1.1 Scope

The focus of this paper is on Security Labeling Rules and the use of emerging artifacts and tools in FHIR to model, express, encode, and communicate them. General introduction to the security labels and the SLS, as well as discussions on the implementation models for the SLS have been covered by various other reports and papers; while a brief overview is provided here, this paper will not provide and in-depth discussion of those topics.

Likewise, CQL and the artifacts of the Clinical Reasoning module of FHIR are also discussed very briefly and to the extent that are used in the proposed architecture by this paper. For a more thorough discussion of these topics, see the respective documentation.

2 CURRENT APPROACH: SIMPLE SECURITY LABELING RULES

While the concept of security labels and the Security Labeling Service has been around for over a decade, implementations of SLS, especially for FHIR, did not emerged until recently in the past few years.

Most current implementations of SLS operate based on a simple model for Security Labeling Rules as a look-up table of clinical codes[2] as shown in **Figure 1**. This table contains the list of qualified clinicals codes (in the form of *system-code* pairs) that identify sensitive clinical conditions.

The SLS labels any resource in which a sensitive code appears with the security labels specified by the corresponding row in the table. In other words, the rules table is essentially a short-hand to represent a set of simple *if-then* rules that indicate that *if* a clinical code appears in the resource *then* a sensitivity label must be assigned. In this model, the SLS operates on a single resource at a time and links between related resources are either disregarded or handled by hard-coded logic.

The simple if-then table is often implemented as a relation (i.e., table) in a relational database (either persisted or in-memory), although sometimes this table is stored as a simple comma-separated file that is read into memory at the time of application start-up.

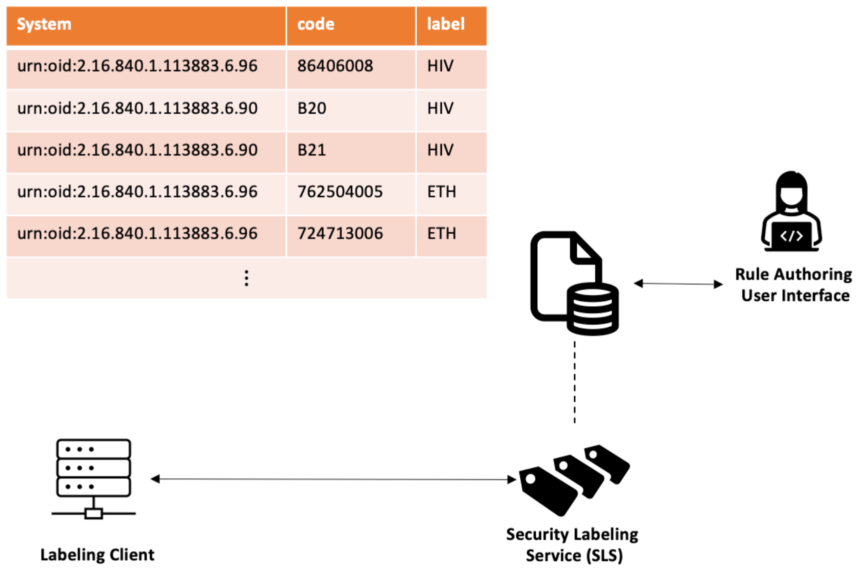
More advanced implementations sometimes use multiple such tables for different types of contexts to encode the rules belonging to different jurisdictions, but the essence of the rules is still the same simple *if-then* logic.

Another variation of this model is when a secondary table is used in a second scan of the resource in which confidentiality labels are determined based on sensitivity labels assigned in the first scan. This second table is essentially a representation of the jurisdictional or organizational policies. For example, a row in the secondary table can indicate that *if* the resource bears a Human Immunodeficiency Virus (HIV) sensitivity label, *then* it has to be assigned a confidentiality label *restricted*. This rule essentially captures a jurisdictional or organizational policy that stipulates *"all HIV-related information is considered restricted."*

Note that in both of these variations, there is an implicit part in the rules which captures the context of the data collection or processing, identifies the applicable jurisdictional policies, and enact the suitable table. This implied logic is in fact part of the logic of the Security Labeling Rules that is not written down or recorded and is implied by the workflow and the application logic of the SLS. As discussed in Sections 2.1 and 2.2 below, this is problematic both for interoperability and configurability of the SLS.

Note that there are sometimes transaction-specific labeling rules, such as assigning handling instructions or purpose of use labels that depend on the specific recipient of the data and the context of the transaction. Such rules are part of the authorization and privacy policies and the SLS has to process the respective policies directly.

[2] This is based on the implementations presented in various security labeling tracks at the HL7 FHIR Connectathons, as well as other proprietary and pilot implementations in which the authors have been involved or of which they are aware.



System	code	label
urn:oid:2.16.840.1.113883.6.96	86406008	HIV
urn:oid:2.16.840.1.113883.6.90	B20	HIV
urn:oid:2.16.840.1.113883.6.90	B21	HIV
urn:oid:2.16.840.1.113883.6.96	762504005	ETH
urn:oid:2.16.840.1.113883.6.96	724713006	ETH
⋮		

Rule Authoring User Interface
Labeling Client
Security Labeling Service (SLS)

Figure 1: Simple Non-Interoperable Security Labeling Rules

2.1 Expressiveness of the Rules

The simple table model presumes an unrealistic simplicity in the Security Labeling Rules by assuming that these rules can be captured by simple *if-then* logic operating on the attributes of a single resource. However, many use-cases require rules with the level of complexity far beyond this simple model. For example:

- Basic logical expressions are often necessary in labeling rules. For example, a conjunction of two different types of medication (each of which is individually innocuous) could be indicative of a sensitive condition.
- Special sensitivity codes regarding adolescent's right to privacy (from their parents) hinge upon the age of the patient with whom a prescription or encounter is associated, at the time of the encounter or prescription.
- Some sensitivity labels depend on the facility where care has taken place, or the type of the practitioner who has provided care. For example, a prescription of a normal pain killer is not sensitive, but the same prescription by a psychiatrist, or at a substance-abuse treatment facility is sensitive.
- Certain information associated with a newborn could lead to inference about sensitive conditions in the mother; expressing such rules requires that the author of the rule be able to assign a sensitivity label based on the codes appearing in another (but related) patient's resources.

Overall, sensitivity of medical information often depends on complex criteria that go far beyond the simple *if-then* model for Security Labeling Rules.

Aside from the complexity of a rule itself, there is usually additional logic around how and when the rule must be applied; for example, the preconditions, triggers, and post-conditions. For example:

- The trigger for a rule, for example, “update the Encounter labels if new Conditions are linked to it.”
- The cascading effects of the rule, “if a Condition is labeled with a sensitivity label, any Encounter that references this Condition should also be labeled with the same sensitivity label.”

In the simple table model for Security Labeling Rules, there are no facilities to express these components, so, the logic has to be either hard-coded or implemented using another rules engine. This means that the rules logic will either be fragmented or implied which has undesirable consequences including the following:

- When parts of the logic of the rules are hard-coded in the implementation of the SLS, changing or updating those parts of the rules logic requires an update to the software. In other words, the SLS will be unable to accommodate certain updates to the rules via configuration and the vendor or the developer team has to be involved by pushing an expensive software update.
- When parts of the logic of the rules are implicit, sharing the rules with other parties becomes extremely difficult if not impossible. This requires that the exchanging parties have a common understanding of the implied logic via offline mechanisms (such as agreements or sharing the same software) in order to ensure that the rules are correctly executed after an exchange.

2.2 Non-Interoperable Rules

In the simple table model, Security Labeling Rules are expressed using a non-standard and non-interoperable format in a databased or a file.

Aside from the rules themselves, there is also no standard set of metadata for rules (e.g., the author, the jurisdiction, the version, etc.) or format and structure to store them.

The lack of standard and interoperable format also inevitably trickles down to the persistence-layer technology (e.g., relational database, simple files, etc.) and the implementers have to devise and use, for example, database schema or file formats that are not interoperable across different implementations.

Lack of a standard convention for expressing rules and metadata makes it infeasible to provide a standard set of rules for a jurisdiction in a way that can be imported and applied by different implementers and vendors with consistent and accurate results.

Note that as discussed in Section 2.1, other than the metadata for the rules, there is also often additional logic around the application of the rules (such as triggers, pre-conditions, post-conditions, etc.) which has to also be stored in proprietary formats, or worse, hard-coded in the SLS application logic.

2.3 Proprietary Rules Management API

In the simple table architecture, the Application Programming Interface (API) for managing and maintaining Security Labeling Rules is either non-existent or proprietary. Such management API is necessary in allowing clients to perform common operations such as retrieving, searching, creating, updating, or deleting the labeling rules.

For example, a client may need to get a list of all the rules belonging to a specific jurisdiction, or all the rules authored by a given user. Updating rules is also a common use-case when flaws are detected in the existing rules or overarching laws and guidelines change. Versioning and tracking of changes are essential in a management interface.

Without a management API, the only way to perform such operations is to use a proprietary user interface that directly connects to the underlying persistence layer and provides rules management functions. This means third-party clients cannot access the rules automatically and rules management can only take place via the user interface and manually, by a human user.

But even if this API is implemented, lack of a standard interface makes interoperability an issue since clients will have to deal with different systems differently and, for example, a batch update to all Security Labeling Services within a state after a change in overarching policy will be challenging due to different proprietary APIs. Lack of an interoperable API also makes it challenging to import, export, or exchange the rules.

2.4 Proprietary Labeling API

In the simple table architecture, the API for security labeling is also proprietary and non-standard. Each implementation of the SLS has to provide its own guidance on the request and response format and structure, so, clients cannot rely on a standard labeling API for communicating with an SLS.

3 CLINICAL QUALITY LANGUAGE (CQL)

Clinical Quality Language (CQL) is a high-level language that enables expressing clinical knowledge targeted at covering Clinical Decision Support (CDS) and Clinical Quality Measurement use-cases. CQL, together with the set of associated tools, provides a conceptual infrastructure for sharing clinical logic in a verifiable and computable way, ready for processing applications including rules execution engines [4].

CQL provides a rich language and a wide variety of conceptual tools to express rules, conditions, queries, and instructions in healthcare workflows and care plans. This expressive power is appropriately aligned with the expressive requirements for Security Labeling Rules, including nuances and complexities discussed in the previous section.

As the next generation standard for health information, FHIR provides full support of CQL. At the conceptual level, CQL is fully incorporated in FHIR and many FHIR resources allow CQL expressions in various attributes; for example, the `Library` [6] and `PlanDefinition` [7] resources. At the computational level, major FHIR implementations directly support processing and evaluating CQL expressions; for example, HL7 Application Programming Interface (HAPI) FHIR, a major open-source implementation of FHIR embeds a CQL evaluation module [8]. Conversely, while CQL is more abstract than FHIR and can support other data models, FHIR is directly supported as one of the major data models in CQL.

While a comprehensive introduction and overview of CQL is beyond the scope of this paper, major components that are directly relevant to expressing Security Labeling Rules are introduced in the rest of this section.[3]

```
library SLS version ‘1’

using FHIR

context Unfiltered

codesystem “SNOMED-CT”: ‘urn:oid:2.16.840.1.113883.6.96’
codesystem “ICD-10-CM”: ‘urn:oid:2.16.840.1.113883.6.90’

code “HIV (SNOMED)”: ‘86406008’ from “SNOMED-CT”
code “HIV (ICD-10)”: ‘B20’ from “ICD-10-CM”

concept “HIV”: { “HIV (SNOMED)”, “ HIV (ICD-10)” } display ‘HIV’

define “HIV-sensitive Conditions”:
   [Condition: “HIV”]

define “HIV-sensitive Encounters”:
   [Encounter]
      with [Condition: “HIV”]
```

Figure 2: A CQL example defining components to isolate HIV-sensitive Conditions and Encounters

[3] Straightforward syntactic and semantic facilities such as logical operators (e.g., AND and OR) are also not discussed. For a detailed discussion of the syntax and semantics refer to the CQL documentation [4].

A running example is shown in Figure 2. This example shows a simple CQL library that defines simple components to isolate HIV-sensitive `Conditions` and `Encounters`. As indicated on line 3, this library is defined based on the FHIR data model. Moreover, line 5 indicates that the library statements are within the *global* context (i.e., *all* resources as opposed to resources associated with a specific patient or practitioner).

3.1 Code Systems, Codes, and Value Sets

CQL enables importing and referencing code systems, individual codes, and value sets. These can be referenced within the logic to construct queries and rules. Lines 7-11 in Figure 2 show examples for importing two code systems and selecting two codes from them, which are later used in the library.

3.2 Concept Definitions

Besides supporting externally-defined value sets, CQL allows defining value sets directly by listing a number of codes via *concepts*. For example, on line 13 in Figure 2, a concept is defined by listing two clinical codes from two different code systems indicating HIV.

3.3 FHIR Resource Queries and Filters

CQL statements allow selecting FHIR resources based on their type and filtering them based on their content. The simplest form of filter is shown in the example on line 16 in Figure 2 which shows a filter based on the value of the `code` attribute. Note that internally-defined concepts (or externally-defined value sets) can be referenced in the filter.

CQL provides a sophisticated set of conceptual tools for filtering which includes traversing resource content to reference specific attributes, as well as `where` clauses, logical operators, and `sort` rules similar to those common in relational database query languages. For example, the rule for filtering conditions can easily restrict the resources to a specific time period, severity, verification status, etc.

3.4 Graph Traversal

CQL provides sophisticated mechanisms for traversing the graph of related resources. A simple example on lines 19-20 in Figure 2 shows how to filter `Encounters` by traversing the related resources graph to find the corresponding `Condition` resource and conduct filtering based on the type of that condition. This is a simple example, but more complex rules can be expressed similarly; for example, filtering based on the type of the facility where an encounter happens (e.g., *"all encounters taking place at a substance abuse treatment facility")* or the role of the practitioner (e.g., *"any encounter with a psychiatrist"*).

4 PROPOSED ARCHITECTURE

Figure 3 depicts the high-level overview of the proposed architecture for authoring, storing, exchanging, and enforcing computable Security Labeling Rules using FHIR. The main features of this architecture are as the following: note that these directly address the issues and shortcomings of the current simple models as discussed in Section 2:

- Using CQL's rich language and model for formulating the rules logic (including triggers, pre-conditions, post-conditions, etc.)
- Using standard FHIR resources, namely PlanDefinition and Library, for storing rules and their associated metadata, without relying on any other proprietary persistence layer.
- Using standard FHIR API and its full features for searching, retrieving, exchanging, creating, updating, deleting, and setting notifications about rules.
- Using a standard API for requesting labeling decisions from the SLS based on the Clinical Decision Support Hooks [10].

The rest of this section discusses the components, interfaces, and their interactions in this architecture.

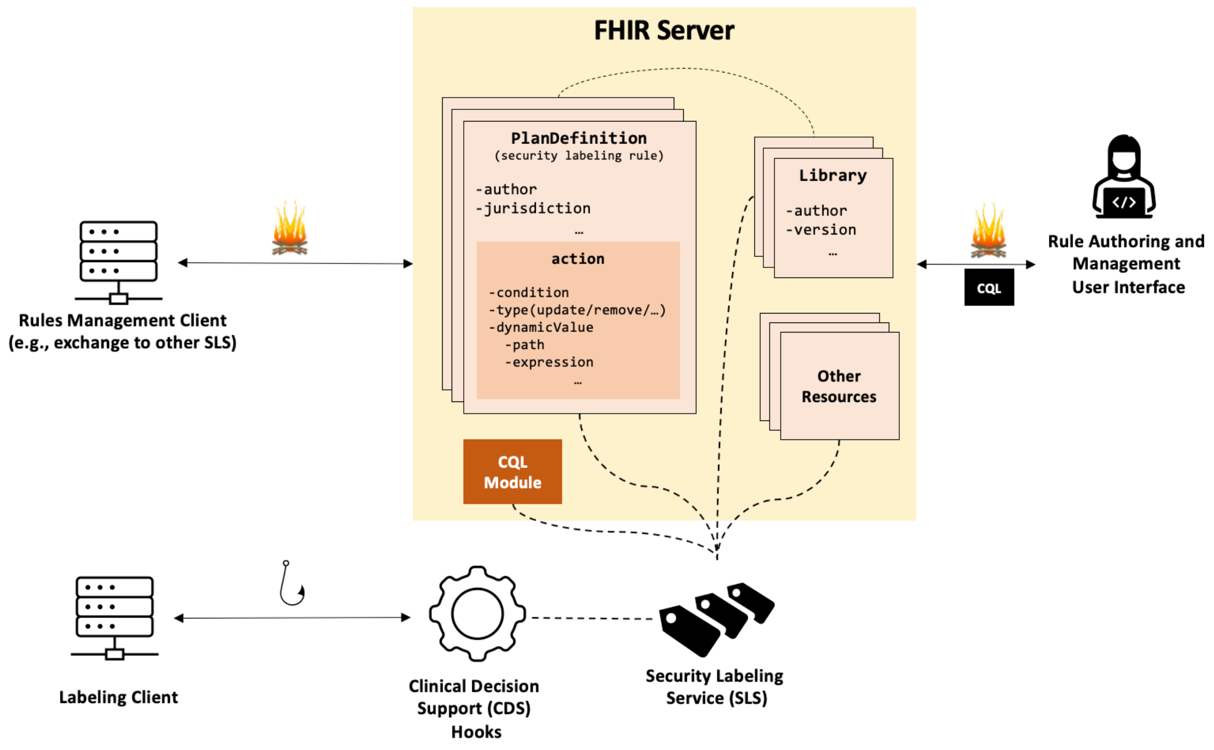


Figure 3: Proposed Architecture

4.1 FHIR Server

FHIR Clinical Reasoning Module [5] defines a number of resources for modeling and encapsulating a variety of clinical knowledge. `PlanDefinition` is one of these resources suitably designed to model (among other things) clinical decision support rules [7]. This resource captures a wide variety of metadata (e.g., version, description, jurisdiction, approval date, effective date, documentation, etc.). Moreover, the resource includes conceptual elements for modeling Security Labeling Rules in a computable way, including a flexible structure wherein a hierarchy of actions and sub-actions can be modeled together with elements that capture the rule logic, such as *triggers*, *conditions*, and inter-action dependencies such as pre- and post-conditions.

CQL is widely supported in modeling the logic of the rules in these attributes. Furthermore, the `Library` resource enables storing standard logical components that could be re-used in expressing the rules logic within `PlanDefinitions`.

With these resources, FHIR provides the required facilities to store, maintain, version, and manage Security Labeling Rules in a standard and interoperable way.

4.2 Rule Authoring and Management User Interface

Authoring, testing, updating, and maintaining Security Labeling Rules requires various types of expertise and skills, ranging from clinical knowledge, terminology expertise, and healthcare information technology. Thus, a user-friendly interface with appropriate user experience design is needed to facilitate the process of rules management.

Since rules are stored in a FHIR server as standard FHIR resources, the graphical user interface for rules management can interface with the persistence layer using standard FHIR API.

Note that it is important to ensure that the user interface is implemented based on a pure FHIR persistence layer without relying on a proprietary database for persisting additional data or metadata. Any additional rules metadata (if not directly supported by the FHIR resources) can be stored in the form of FHIR extensions on the `PlanDefinition` resources.

4.3 Rules Management API

Using FHIR for storing rules means that the FHIR API can be used as a standard rules management API, enabling searching, retrieving, creating, updating, and deleting rules. Moreover, access to Security Labeling Rules will be available via other FHIR mechanisms such as Subscription [9], which enables clients to set notifications based on changes in rules or creation of new ones.

The rules management API also enables exchanging and synchronizing rules with other systems including other SLS instances (e.g., from other organizations) to ensure a seamless distribution of rules across a group of systems.

4.4 Security Labeling Service (SLS)

By parsing and understanding Security Labeling Rules, the SLS provides the service of assigning security labels. The SLS can interact with the FHIR server using the standard FHIR API to search and retrieve applicable Security Labeling Rules. Note that for computational efficiency, the SLS should cache the Security Labeling Rules so that it does not need to search for and retrieve rules using the (costly) API calls and parse the retrieved resources for every instance of labeling; this will be further discussed in Section 5.2.

Processing rules and understanding their semantics requires that the SLS can consume and evaluate CQL expressions. This requires a CQL module which may reside inside the FHIR server itself or within the SLS.

The SLS can work internally as part of the FHIR server to automate the process of labeling resources. Alternatively, the SLS can provide a standard labeling API so that different clients (residing inside or outside the FHIR server) can request labeling of given resources.

4.5 Labeling API

The Clinical Decision Support (CDS) Hooks specification is a FHIR-based specification that defines a standard API for querying a CDS system [10]. This API defines the format in which the request can be sent to the CDS and the structure and format of the response.

The CDS Hooks API is aligned with the structure in which clinical decision rules are expressed using `PlanDefinition` resources, and therefore, it provides a suitable specification for the SLS labeling API.

Figure 4 shows a schematic representation of an SLS labeling request and response based on the CDS Hooks specifications. The data structures will be discussed in the rest of this section.

4.5.1 SLS Request

The following is a sketch of the SLS request based on the CDS Hooks specifications, as also shown in **Figure 4**:

- The request declares that it intends to query the SLS hook by specifying the hook name, for example, assign-security-labels.
- The metadata about the transaction context (e.g., purpose of use, recipient’s identifier, etc.) is included in the context attribute of the hook request.
- The content of the resource to be labeled is included in the prefetch attribute of the request.

4.5.2 SLS Response

The following is a sketch of the SLS response based on the CDS Hooks specifications, as shown in **Figure 4**:

- Since the CDS response is intended for a software client, it includes no cards (i.e., an empty `card` objects).
- An array of `action` objects is provided as the value of the `systemActions` attribute.
- The `action` objects of type `update` indicate the assignment of security labels. The parameter provides the labeled version of the resource. Note that this may be inefficient for large resources (e.g., large bundles) and can be improved as will be discussed in Section 5.1.
- Optionally, `action` objects can instruct the client, for example the Privacy Preserving Services (PPS), to redact a resource from a bundle using the `delete` action type.

Other action types can be defined as extensions to the current CDS Hooks specifications to accommodate other SLS instructions if necessary.

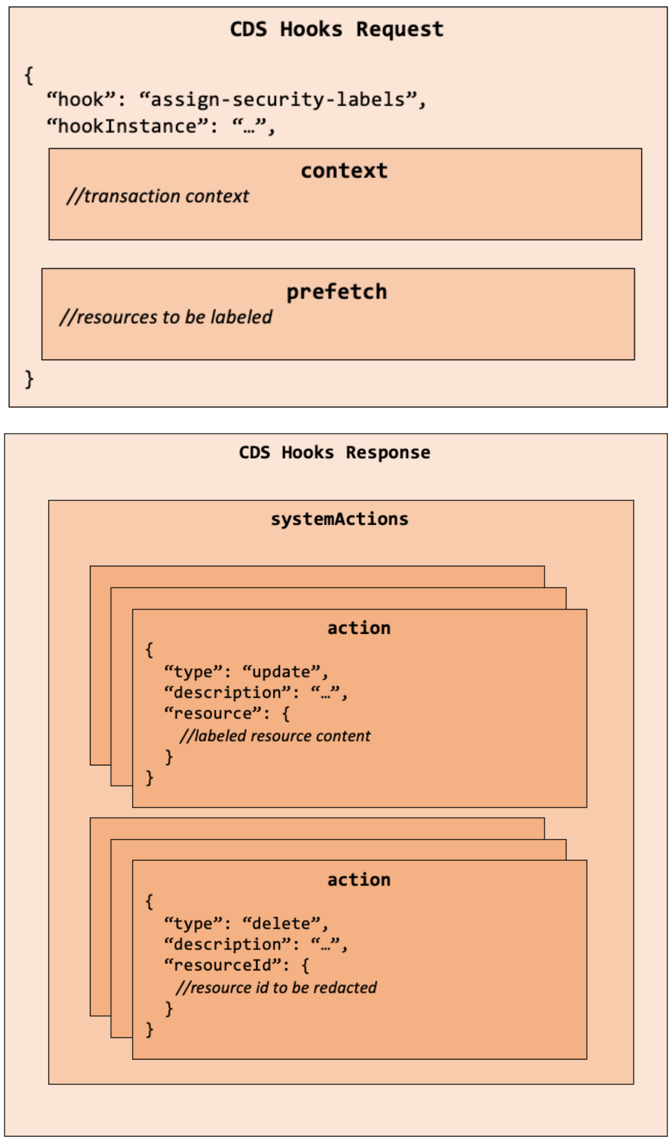


Figure 4: Sample SLS request and response based on the CDS hooks API format

5 DISCUSSION

This section provides some discussion points and noteworthy observations for implementers.

5.1 Labeling Patch

The CDS Hooks API specifications currently require that the entire resource be included in the response with an update action and do not provide a mechanism to only specify the updated portion of the resource –in this case the security labels. This is inefficient especially for large resources, such as large bundles.

A simple update to the specification can allow a `patch` action type to allow the SLS to only specify the updated parts of the resource. The FHIRPath Patch specifications defines the mechanisms for specifying such partial updates [11].

5.2 SLS Operational Model

Reading and parsing Security Labeling Rules in the form of FHIR resources using the FHIR API is quite costly for applications that call for labeling of millions of resources, for example, when doing a bulk import. Practically, the overhead of making API calls to search and retrieve rules and the computational overhead of the subsequent parsing of the rules is too high for individual labeling calls, especially considering that change in Security Labeling Rules is often infrequent.

It is important for implementers to note that relying on standard API and FHIR resources does not mean that the SLS should internally rely on the same data structures. The SLS should use suitable strategies such as caching rules and converting them into efficient internal data structures to provide a scalable service with reliable performance.

5.3 Asynchronous SLS API

The architecture in this paper proposes using the CDS Hooks API as the basis for interacting with the SLS. The CDS Hooks API is currently a synchronous API which requires the SLS to respond in real time to labeling requests.

This is generally a reasonable expectation for labeling in FHIR, however, there might be special cases where FHIR resources are constructed in ways that require a deeper inspection and more computationally heavy processing in the labeling process. For example, if FHIR resources widely use unstructured texts and attachments without recording that information in the structured parts of the resources, a Natural Language Processing (NLP) module is necessary to ensure all the implicit clinical codes in the unstructured text portions of the resource are identified and considered in the labeling process.

Heavy computations like NLP, however, could make it impossible for the SLS to respond synchronously and may call for an asynchronous model in which the client submits the request and receives the response later. Since the CDS Hooks API does not support asynchronous calls, this is not currently supported and requires defining an extension.

5.4 FHIR Implementation Guide

Using `PlanDefinition` provides a basic standard structure for expressing, storing, and communicating Security Labeling Rules. In order to achieve a more advanced level of interoperability which guarantees consistent and correct interpretation of the computable parts of the rules, a profile of the resource is necessary, in which further details such as the nuances of modeling the conditions, actions, triggers, and data requirements are narrowed down. Such details can be formalized using a FHIR Implementation Guide that provides the details and examples to help implementers correctly create, consume, and interpret Security Labeling Rules in the form of `PlanDefinition` resources.

6 REFERENCES

[1] FHIR Data Segmentation for Privacy Implementation Guide v0.2.0, Value-Sets Summary. http://build.fhir.org/ig/HL7/fhir-security-label-ds4p/branches/master/spec.html#value-sets-summary

[2] FHIR Data Segmentation for Privacy Implementation Guide v0.2.0, Inline Labeling. http://build.fhir.org/ig/HL7/fhir-security-label-ds4p/branches/master/inline.html

[3] FHIRPath. https://hl7.org/fhirpath

[4] Clinical Quality Language (CQL). https://cql.hl7.org

[5] FHIR Clinical Reasoning Module. http://www.hl7.org/fhir/clinicalreasoning-module.html

[6] FHIR Library Resource. http://www.hl7.org/fhir/library.html

[7] FHIR PlanDefinition Resource. https://www.hl7.org/fhir/plandefinition.html

[8] HAPI FHIR, CQL Module. https://hapifhir.io/hapi-fhir/docs/server_jpa_cql/cql.html

[9] FHIR Subscription. https://www.hl7.org/fhir/subscription.html

[10] Clinical Decision Support (CDS) Hooks. https://cds-hooks.org

[11] FHIRPath Patch. https://www.hl7.org/fhir/fhirpatch.html

7 ACRONYMS

API	Application Programming Interface
CDS	Clinical Decision Support
CQL	Clinical Quality Language
DS4P	Data Segmentation for Privacy
EHR	Electronic Health Record
FHIR	Fast Healthcare Interoperability Resources
HAPI	HL7 Application Programming Interface
HIV	Human Immunodeficiency Virus
HL7	Health Level Seven
NLP	Natural Language Processing
PPS	Privacy-Preserving Service
SLS	Security Labeling Service
SNOMED-CT	Systematized Nomenclature of Medicine Clinical Terms