Serdica Journal of Computing 16(1), 2022, pp. 39-56, 10.55630/sjc.2022.16.39-56
Published by Institute of Mathematics and Informatics, Bulgarian Academy of Sciences

CWE Ontology

Vladimir Dimitrov

Faculty of Mathematics and Informatics,
Sofia University "St. Kliment Ohridski", Bulgaria
cht@fmi.uni-sofia.bg

Abstract

CWE is a community-supported database of known weaknesses. It is under permanent update and development. MITRE Corporation hosts this database. It consists of several viewpoints structured at several abstract levels.

CWE database is the base of CWE ontology. It redefines weaknesses in terms of the Semantic Web. This ontology is a part of ontology ecosystem developed to capture cybersecurity knowledge on known vulnerabilities, weaknesses, and attacks patterns.

CWE ontology classifies CVE/NVD vulnerabilities. It is useful for research and investigation on new vulnerabilities and weaknesses using reasoners. In addition, CWE is useful for cybersecurity incident forensic investigations, software acquisition and development.

Keywords: Cybersecurity, Weakness, Semantic Web, Ontology

ACM 2012 CCS Concepts: Computing methodologies → Artificial intelligence → Knowledge representation and reasoning → Ontology engineering; Security and privacy

Mathematics Subject Classification 2020: 68-04, 68M25, 68T30, 68T35, 68U35

Received: December 4, 2022, *Accepted:* January 10, 2023, *Published:* February 28, 2023

Citation: Vladimir Dimitrov, CWE Ontology, Serdica Journal of Computing 16(1), 2022, pp. 39-56, https://doi.org/10.55630/sjc.2022.16.39-56

1 Repositories and motivation

CWE (Common Weakness Enumeration) is a list of weaknesses maintained by MITRE Corporation [1]. CWE weaknesses are software and/or hardware vulnerability types. CWE is under development by the public cybersecurity community organized in CWE/CAPEC special interest groups (SIGs) and working groups (WGs).

CWE weaknesses classify CVE (Common Vulnerabilities and Exposures) [2] and NVD (National Vulnerability Database) [3] vulnerabilities. Particularly, the view "CWE-1003: Weaknesses for Simplified Mapping of Published Vulnerabilities" classifies NVD.

The difference between CVE and NVD is that in CVE vulnerabilities are under investigation, while NIST have thoroughly analyzed NVD vulnerabilities. NIST team has enriched the information in NVD vulnerabilities.

CVE is a kind of vulnerability research database, while NVD is ready-to-use. MITRE Corporation databases (CVE, CWE, and CAPEC) are evolving, while the selected view of vulnerabilities in NVD is relatively stable. However, NIST officially does not support the weaknesses in the Security Content Automation Protocol (SCAP) [4].

CAPEC (Common Attack Pattern Enumeration and Classification) [5] is a database of known attack patterns.

CVE/NVD, CWE, and CAPEC reference each other. They are cornerstones of the cybersecurity ecosystem of knowledge about revealed vulnerabilities, weaknesses, and attack patterns. XML and/or JSON documents code this knowledge.

The current version of the CWE Weaknesses Catalog is available with no charge from [1]. It is an XML document. The catalog scheme and project documentation are available from the same site.

NVD vulnerabilities are simply a list. However, several taxonomies (views) structure CWE weaknesses. Some of these views have graph structure.

NVD vulnerabilities do not contain mitigation/prevention recommendations. This information is available in CWE weakness.

CWE weakness can participate in several taxonomies; even some CWE weaknesses can participate several times in the same taxonomy. Further, in the taxonomy, the abstraction levels of the weakness ancestors contain more information about it.

Let us consider the following use case: the cybersecurity expert tries to protect a system possessing specific vulnerability. He/she has to navigate in CWE to find all weaknesses in all taxonomies related to this vulnerability.

It is possible for this navigation to be done using traditional (relational) databases and SQL. In that case, the expert has to know the taxonomy structures to formulate adequate queries. However, OWL ontologies and SPARQL can automate this process.

How OWL ontologies and SPARQL can facilitate navigation automatization? In that case, the user, to formulate SPARQL queries, has to know only the main taxonomies and basic relationships structuring them.

An analogy between these two approaches is as to use procedural programming language for implementation of an object-oriented system – it is possible, but difficult. Relational database is suitable to store data and information (processed data suitable for decision support). OWL is a notation for knowledge representation. The knowledge is extraction of principles, models, and relationships from the data (information). OWL represents knowledge in RDF graphs. SPARQL navigates in RDF graphs.

Finally, it is clear, that to require from a cybersecurity expert skills and experience in OWL ontologies and SPARQL is not realistic. Cybersecurity expert systems can cover the internal knowledge representation. These expert systems must have simple intuitive interfaces adequate to support cybersecurity expert activities.

2 CWE catalog structure

CWE catalog consists of `Views`, `Weaknesses`, `Categories` and `External References`.

`View` is a viewpoint on the weaknesses in the catalog. There are three view types: graph, explicit slice, and implicit slice.

The first two types of views directly reference its weaknesses via `Members` element. `Members` element references categories and weaknesses that are included in the view.

The third view type defines its content via `Filter` element that contains an XSL query string filtering view catalog items. This type of views does not reference directly its weaknesses – the query specified in the `Filter` element references view weaknesses.

The slice contains simply a list of weaknesses. The graph structures the view weaknesses at several abstraction levels. The graph view organizes the weaknesses into a taxonomy – possibly hierarchical.

There are some problems with the catalog schema concerning the membership. First, it is possible for a view to contain other views, but there is no

such instance in the catalog content. Second, the category can contain views, again there is no such instance in the catalog content. The same applies to the categories – the category can contain a category. Third, if a view includes categories and weaknesses, then these categories and weaknesses have to be from the same view, but the schema does not contain such a restriction. Moreover, views associate categories with themselves, but the categories can associate its weaknesses with other views. However, the catalog content does not contain such deviations – categories belong to one view and associate its weaknesses with the same view.

Solution of above-mentioned problems is via the catalog content, i.e. the catalog content determines the ontology specification.

`Category` contains catalog entries that share common characteristics. The categories can refer to other taxonomies (external to the ecosystem).

`Weakness` is the basic catalog (therefore view) entry. Their abstraction levels are `Pillar`, `Class`, `Base`, and `Variant`; by structure `Compound` are `Chain`, `Composite`, and `Simple`.

`Pillar` is the most abstract weakness. It is a theme unifying its weaknesses. However, it is not `Category` because it is a weakness (`Category` is a set of weaknesses), but is not itself a weakness. The pillar is an abstract weakness.

`Class` is an abstract weakness. It does not depend on specific language and/or technology, but is more specific than a pillar. The class description is in one or two of the following three dimensions: behavior, property, and resource.

`Base` is more specific than the class. It does not depend on specific language and/or technology, but contains enough details about methods to detected and/or avoid them.

`Variant` is usually associated with a specific language and/or technology. The variant description is in terms of three to five of the following dimensions: behavior, property, technology, language, and resource.

`Compound` weakness is an aggregate of several other weaknesses, currently it is `Chain` or `Composite`.

The weakness structure is `Simple`, `Chain` or `Composite`. The simple weakness does not depend on the other weaknesses. The composite weakness is a set of weaknesses that must be present at the same time to produce an exploitable vulnerability. `Chain` is like `Composite`, but there is some order among participating weaknesses.

The nature of the relationships between weaknesses can be:

- `ChildOf` – the link is to a weakness from a higher abstraction level.

- `ParentOf` is inverse of `ChildOf`. In fact, the catalog contains only such links.
- `StartsWith` – the link points to the beginning a named chain – to the first weakness. There are also unnamed chains in which only the next two types of links are applicable.
- `CanFollow` – the weakness can follow in the chain the weakness in which the link points. Both named and unnamed chains use these links.
- `CanPrecede` is opposite of `CanFollow`. There are only links of this kind in the catalog content.
- `RequiredBy` – the link points to the composite weakness to which it belongs.
- `Requires` – is opposite of `RequiredBy`. There are only links of this kind in the catalog content. Only the composites have names.
- `CanAlsoBe` links to a weakness that is almost like the source. This is not inverse link, i.e. the opposite is not valid.
- `PeerOf` links the weakness with another weakness that is similar to it. However, the link is not of any kind listed above.

The weaknesses can be in languages, operating systems, architectures, and technologies.

The other important information presented in the weaknesses are modes of introduction, exploitation factors, likelihood of exploit, common consequences, impact on security elements, detection methods, potential mitigations, examples (including references to CVEs), functional areas, affected resources and related attack patterns (with references to CAPEC). CWE schema contains more detailed and precise information for these elements.

3 CWE ontology

CWE XSD schema is the base of CWE ontology. Below is an investigation on the presentation of CWE data elements and their attributes.

Some CWE elements and attributes have only descriptive content that is not a formal knowledge. Therefore, annotations represent them, since they do not participate in the reasoning process.

In CWE schema, XML element content represents both objects and relationships. This complicates and obscures the fact presentation. The standard approach for representing relationships in XML is via element attributes. However, in CWE catalog, it is not the case; even some relationships are XPath queries. The ecosystem ontology relationships have to be object properties.

The ontology classes, objects, and data properties must have clear representation. For example, relationships must not be queries – the reasoners do not execute queries. This means, the ontology individuals and properties must not be XPath queries.

CWE catalog is a set of views. Each view contains categories and weaknesses. The last ones can belong to several views. The view defines the relationships between views, categories, and weaknesses, i.e. the views bound all relationships. The ontology follows this vision.

Figure 1 shows CWE ontology structure.

3.1 Views

For each view, the ontology maintains a separate name space. For example, the view "CWE-1000" has the name space http://www.semanticweb.org/cht_c/cwe-1000#.

`Status` and `Type` attributes factorize the views. The leading generalization is by the type. The status is applicable to the weaknesses and categories as well.

By the leading specialization `Type`, the `View` class has three subclasses: `Explicit`, `Graph`, and `Implicit`. It is possible, some view to have no type now. Generally, the class `View` is a union without intersections of its three subclasses.

On the other hand, each catalog item (view, category, and weakness) can be in any of the states defined by Status. The class `Status` has subclasses: `Deprecated`, `Draft`, `Incomplete`, `Obsolete`, `Stable`, and `Usable`. `Status` is a union without intersection of its subclasses. It is possible a `Status` individual not to be member of `State` subclasses, for the moment.

In addition to the above listed subclasses, `Status` has subclasses `View`, `Category`, and `Weakness` (a union without intersections). This means that every view, category, and weakness is also a `Status` individual, i.e. it has a state. In fact, this is the factorization by the status of views, categories, and weaknesses. Although not set, but the reasoners recognize views, categories, and weaknesses as subclasses of `Status`.

In object-oriented design terms, the relationship among views, categories, and weaknesses is a factorization (inheritance) by aggregation. `Status` class is

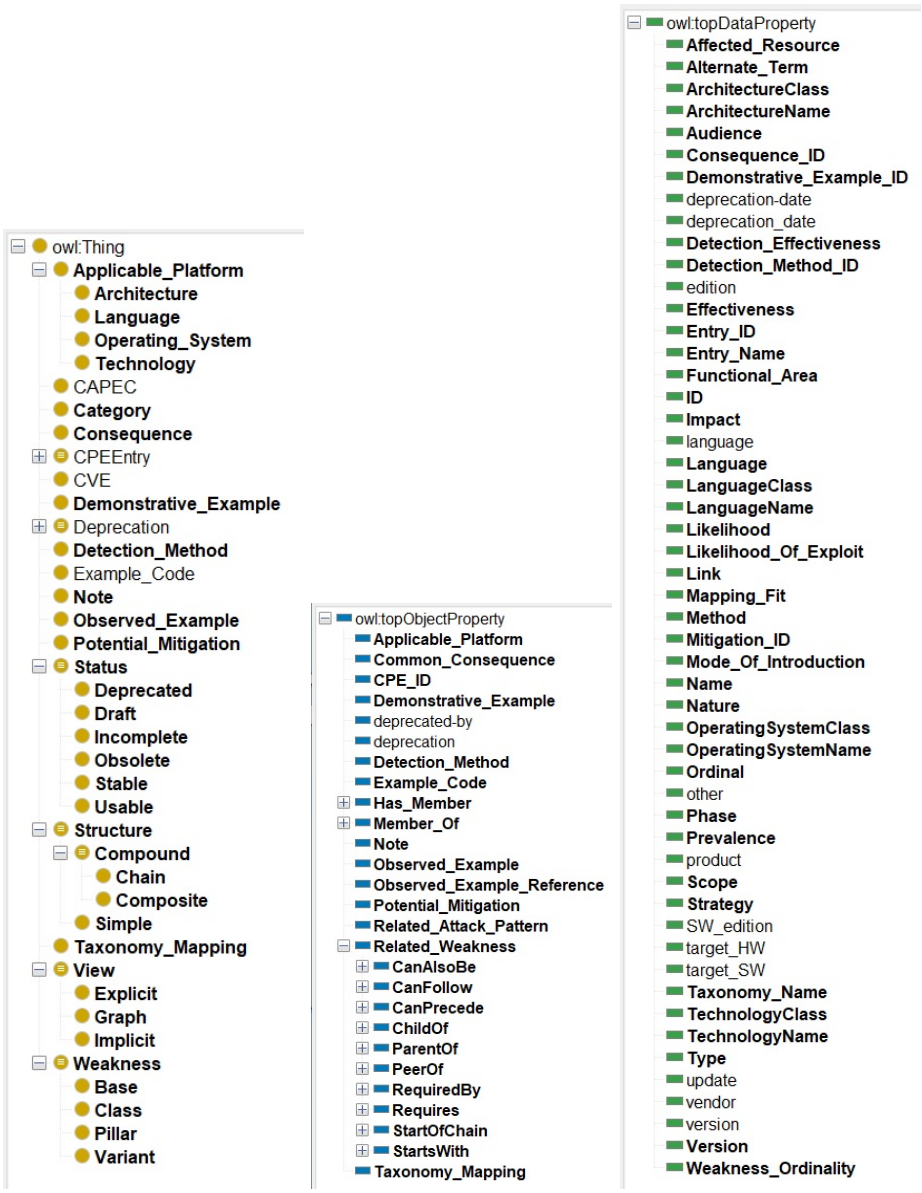

Figure 1: CWE ontology classes, object properties and data properties.

also an aggregate of its subclasses `Deprecated`, `Draft`, `Incomplete`, `Obsolete`, `Stable`, and `Usable`. OWL merges both classification concepts from object-oriented design (inheritance and classification by aggregation). In OWL, the reasoners recognize factorization by aggregation and preserve individuality by turning aggregate inheritance into true inheritance.

The object-oriented design does not recommend creating subclasses without their own properties. Instead, it suggests creating in the parent class attributes with enumerations by the names of the subclasses. In the ontology, however, the opposite approach is applied.

In object-oriented design, maintenance of a class is more expensive than the maintenance of an attribute. In Semantic Web, however, the role of the class is only in the classification process. In OWL, data properties and object properties are reasoning constraints. Reasoning in OWL easier maintains classes than properties. This is the motivation for the application of the opposite approach to the object-oriented design one.

The view has `Name`, which is represented as a data property (a character string); an identifier (number) – `ID`, which is a data property (a positive integer), and `Audience` is also data property, but its value is an enumeration according to the XSD schema. `Name` and `ID` are data properties for categories and weaknesses as well.

`ID` attribute forms the individual IRIs (Internationalized Resource Identifier) of the views, categories, and weaknesses. `ID` forms the weakness IRI – `ID` is preceded by "CWE-". For example, the view with `ID` "1000" has IRI individual with "CWE-1000".

The object properties `Has_Member` and `Member_Of` are inverse. `Has_Member` links the view to its members and `Member_Of` links the view members to the view. Due to the nature of the catalog, `Has_Member` can associate the view with views, categories, and weaknesses, i.e. it is possible a view to have another view as a member.

On the other hand, the categories also use `Has_Member` and `Member_Of`. `RelationshipsType` has two forms for `Has_Member` and for `Member_Of` elements. In the first form, there are one or more `Member_Of` elements and zero or more `Has_Member` elements. The second form is one or more `Has_Member` elements.

From the text in `Member_Of` description, it is clear that when used, the weakness links to its category and to its view, i.e. category and view defines the relationship. In other words, the view references its categories; this is the first variant of `RelationshipsType` type. The remaining `Has_Member` elements in this variant associate the category with its members, which as indicated could be other categories or weaknesses, but not views.

There are no `Member_Of` items in the catalog. Only the second variant of `RelationshipsType` is available in the categories, i.e. one or more `Has_Member` elements.

The interpretation of the second form applies to views and categories. The catalog content determines that a category member can be a category or a weakness, but not a view.

SHACL can control the restriction that categories cannot have views as members in CWE. The ontology is for reasoning but not for constraints on RDF.

`Has_Member` and `Member_Of` elements are of type `MemberType`, which has two required attributes `CWE_ID` and `View_ID`. In fact, `View_ID` specifies to which view the link belongs, and `CWE_ID` can point to a view, category, or weakness.

As defined in the ontology, the object properties `Has_Member` and `Member_Of` do not have associations with any view. This is where view namespaces come in handy. For example, for the view CWE-1178 name space is:
`Prefix: cwe-1178: <http://www.semanticweb.org/cht_c/cwe-1178#>`
For this space (view) are defined the object properties "cwe-1178:Member_Of" and "cwe-1178:Has_Member", which are subproperties of `Member_Of` and `Has_Member` and properties, respectively.

Thus, `Has_Member` and `Member_Of` properties are associated to the corresponding view namespace via its subproperties. In that sense, `Has_Member` and `Member_Of` are "abstract" properties.

For example, the category "CWE-1179", which has members according to the view "CWE-1178", references its members through the corresponding subproperties of the above mentioned properties "cwe-1178:Has_Member" and "cwe -1178:Member_Of".

Implicit views use `Filter` attribute in the catalog. This attribute contains XPath query. In fact, the content of this view type is dynamic – the query execution generates the view members. There are no explicit members assigned with `Members` element in the catalog content.

This approach does not work in OWL – ontology does not compute dynamically its content. OWL graph is static. Therefore, the ontology must include computed ontology content. `Filter` attribute becomes a view annotation.

The last group of `References`, `Notes` and `Content_History` elements has the same meaning and content as in the categories and weaknesses.

`References` element consists of `Reference` elements and is entirely descriptive. These are references to external sources in "REF-n" format. They may have a section description, which is an identifier type.

The ontology has `Reference` annotation. It includes the information from all elements with the same name. The reasoning does not use information from this element.

`Notes` element consists of `Note` elements, which are structured text with a required `Type` attribute.

The class `Note` represents the `Note` elements. The views (categories and weaknesses) point to individuals of this class. The notes are additional detailed information about the views.

Each individual `Note` has an IRI formed by the name of the weakness, followed by the text "Note" and a sequence number starting from zero. For example, "CWE-103" has two notes and the corresponding `Note` individuals are "CWE-103_Note0" and "CWE-103_Note1".

`Type` attribute classifies the notes, although they are descriptive, the reasoning can use them.

The annotation `Note_Description` represents the note description.

`Content_History` element is purely descriptive. The catalog maintenance process uses them. The annotation `Content_History` represents it in the ontology. Its content follows the structure of the element.

Annotations can have only one domain. Setting the annotation domain has no effect on the reasoning, nor the annotations themselves. However, specifying an annotation domain aligns the ontology description where possible.

For example, the catalog elements use `Description` element in different contexts. The ontology defines different annotations with the class name followed by `Description` to differentiate the contexts. In this example, it is `Note_Description`.

3.2 Categories

`Category` class has no subclasses of its own, but like `View` is a subclass of `Status` for the reasoners.

`Name` and `ID` data properties have the same purpose as in views. `ID` is a redundant property; it forms individual IRIs for views, categories, and weaknesses.

An annotation with the same name represents `Summary` element – it is an extended description of the category.

`Relationships` element is similar to that one in `Members` element of the views – with the same meaning and content that have discussed for the views.

`References`, `Notes`, and `Content_History` elements have the same meaning and content as in the views.

`Taxonomy_Mappings` element represents a set of `TaxonomyMapping` elements. They link CWE elements to other taxonomies. `TaxonomyMapping` class represents these mappings. The object property with the same name links the category to the other taxonomies entries.

The same `Taxonomy_Mappings` element is available for weaknesses with the same meaning and content.

IRIs of taxonomy mapping individuals is similar to the formation of `Note` IRIs, but instead of a "Note" string, a "Taxonomy_Mapping" string is used.

`Taxonomy_Mapping` individuals have `Taxonomy_Name` data property that specifies the target taxonomy. The other data properties are `Entry_ID`, `Entry_Name`, and `Mapping_Fit`. `Entry_ID` and `Entry_Name` specify exactly in which element of the target taxonomy, the category (weakness) maps. In addition, `Mapping_Fit` describes how suitable this mapping is. The last is an enumeration.

3.3 Weaknesses

`Weakness` class has its own subclasses. These are `Pillar`, `Class`, `Base`, and `Variant`. `Weakness` class factorization is by the abstraction level (the attribute `Abstract`). `Weakness` is a union without intersection of its subclasses.

The weaknesses do not intersect with views and categories.

The class `Weakness` is equivalent to `Structure` class. This simplifies the specification.

The structures are of two types: simple and compound, which are the subclasses `Simple` and `Compound`.

The class `Structure` is a union without intersection of its subclasses.

On the other hand, the compounds (`Compound` class) are chains and composites, which are the subclasses `Chain` and `Composite`.

The compound structure is a union without intersection of its subclasses.

In weaknesses, `ID` and `Name` attributes are with the same meaning and content as in the views.

`Status` attribute has the same presentation, as in `View`, i.e. `Weakness` is a subclass of `Status`.

`Weakness_Description` annotation is the weakness description. Similar presentation has the extended description with `Extended_Description` annotation.

`Related_Weaknesses` element represents the relationships between weaknesses in CWE XSD schema. It consists of subelements `Related_Weakness`.

The last has five attributes Nature, `CWE_ID`, `View_ID`, `Chain_ID`, and `Ordinal`. The first three attributes are required.

`Related_Weakness` is an object property describing the relationships between weaknesses in the ontology.

`Nature` attribute specifies the type of relationship in CWE XSD schema. This attribute defines nine object subproperties: `ChildOf`, `ParentOf`, `Starts With`, `CanFollow`, `CanPrecede`, `RequiredBy`, `Requires`, `CanAlsoBe`, and `PeerOf`. Additional is `StartOfChain` object subproperty symmetric with `StartsWith`. Inverse pairs are `ChildOf` – `ParentOf`, `StartsWith` – `StartOfChain`, `CanFollow` – `CanPrecede`, and `RequiredBy` – `Requires`. `CanAlsoBe` and `PeerOf` are neither inverse nor reflexive.

The relationships between weaknesses are in the view context. For each view, there are subproperties of the above ten subproperties in the view namespace. For example, the view "CWE-1026" has the object property "cwe-1026:Requires" in its name space.

`Ordinal` attribute specifies whether the relationship is primary or not. This attribute has only value `Primary`. There is only one primary relationship for the weakness in the view. To model this attribute for each of the ten object properties there are object subproperties with the same name but with the extension "-Primary". For example, the view "CWE-1026" has the object property "cwe-1026:Requires-Primary".

The above mentioned restrictions are kind of functional dependencies in database terms. They are mandatory and only SHACL can control them – the closed world assumption.

The case of named chains is more special. There is a single view in the catalog containing the named chains. This is CWE-709. The other views have chains but without names.

From the combination of (`Nature`, `CWE_ID`, and `View_ID`) with `Chain_ID`, one would expect that the view should be the same for all the chain members following the links, but this is not the case in the catalog content.

The chain, more precisely the chain name, points with `StartsWith` to the first weakness of the chain in the view CWE-709. For example, the chain "CWE-680" has the relationship triple (`StartsWith`, CWE-190, CWE-709) and "Chain_ID="680"".

The chain starting from the first weakness to the end looks like unnamed chain from another view. The property `CanPreceed` links all weaknesses in the chain. `CanPreceed` uses `Chain_ID` pointing to the chain name, albeit from another view. In CWE-190 from the example, the triple to the next element in the chain is (`CanPreceed`, CWE-119, CWE-1000), but Chain_ID="680" – here

the view is CWE-1000.

Without `Chain_ID` in `CanPreceed` / `CanFollow`, the result is an unnamed chain.

In fact, there are no `CanFollow` relationships in the catalog content.

The view name and the `StartsWith` and `CanPreceed` / `CanFollow` relationships must be from the same view, namely that of the chain name, but this is not the case in the catalog content.

The relationships in CWE XSD schema are always associated with a view, but there is an obvious violation of this principle in the case of named chains.

To correct above-mentioned violation, the ontology solution is:

1. `StartsWith` and `StartOfChain` subproperties are only in the CWE-709 view, accordingly, and their Primary versions.

2. All weaknesses participating in the named chain are members of view CWE-709.

This means to ignore the unnamed chain in the other views. In the example with chain 680, the name belongs to view 709, but the chain members belong to view 1000. In the ontology, the name, and all chain member belong to view 709, and the view 1000 has no chain without name.

In the formation of the chains, there is another problem; it is possible the chain to be a tree. CWE XSD schema allow this. At this stage, in the catalog content the chains are chains but not trees. SHACL can check this as requirement.

CWE XSD schema relates some weaknesses with other weaknesses. `Weakness_Ordinality` data property describes it. It is not clear what these relationships are and in what context. Several relationships of this kind can exist. Eventually, in `Weakness_Ordinality_Description` annotation some information for this property is available.

`Applicable_Platform` class and an object property with the same name describe the applicable platforms, i.e. to which platforms the weakness is applicable.

This class has four subclasses `Language`, `Operating_System`, `Architecture`, and `Technology`. `Applicable_Platform` element factorizes these subclasses.

The only common attribute of these four subelements is `Prevalence`. `Name` and `Class` attributes have different contents for these subelements. Therefore, the last have different names as data properties. These data properties are enumerations.

The operating systems have two additional attributes: `Version` and `CPE_ID`. A data property version represents the version. `CPE_ID` is an object property that points to a CPE individual.

Additional details are not directly related to the weakness, but still containing information relevant to it as a structured text, are annotations. `Background_Detail` represents separately each additional detail. They are using HTML formatting. In the ontology, a literal represents this text, but its visualization in Protégé is in source code, i.e. it does not render like in a browser. Possibly, in a different environment it could render differently.

The alternative terms add some description to the weakness. Data property `Alternate_Term` and its description `Alternate_Term_Description` annotation represent them.

The different phases of SDLC can introduce weaknesses. The data property `Mode_Of_Introduction` (enumeration type) reflects this. `Mode_Of_Introduction_Note` may annotate this data property.

The exploitation factors are descriptions; `Exploitation_Factor` annotation represents them.

`Likelihood_Of_Exploit` data property represents the likelihood of the vulnerability exploit. It is an enumeration.

The exploitation of a weakness (vulnerability) can have multiple consequences. `Consequence` class represents the consequences. `Common_Consequence` object property points to its individuals.

The consequence individuals have IRIs formed in the familiar way, but using the string "_Consequence". They can have `Consequence_ID` data property, which is an identifier for internal use by CWE team.

The scope of the consequences is a set with Scope data property, which is an enumeration.

`Impact` data property represents the technical impact of the exploited weakness – it is an enumeration.

`Likelihood` data property for consequences has the same meaning and content as `Likelihood_Of_Exploit` presented for the weaknesses, but here it refers to the consequences. `Consequence_Note` annotates the consequence's individuals.

`Detection_Method` class and an object property with the same name represent the methods for weakness detection. Individuals of this class form IRI in the familiar way, but with the string is "_Detection_Method".

The detection methods have `Detection_Method_ID` data property. CWE team uses it.

The detection methods have a data property defining the method itself (`Method`) and its effectiveness (`Detection_Effectiveness`) – a data property. Both data properties are enumerations.

`Detection_Method_Description` and `Effectiveness_Note` annotate the detection methods.

`Potential_Mitigation` class and an object property with the same name represent the potential mitigations for a weakness. Individuals have IRIs formed in the familiar way, but with the string is "_Potential_Mitigation".

`Mitigation_ID` is a data property. CWE team uses it.

The application phase (`Phase`), mitigation strategy (`Strategy`) and its effectiveness (`Effectiveness`) are data properties of enumerations.

`Potential_Mitigation_Description` annotation and the effectiveness `Effectiveness_Note` annotation describe the potential mitigations. The description is a required element.

The demo examples present code containing the described weakness. `Demonstrative_Example` class and an object property with the same name associate the weakness with the example. Individuals have IRIs formed in the familiar way, but with the string "_Demonstrative_Example".

`Demonstrative_Example_ID` is a data property. It is only for use by CWE team to distinguish repeated examples.

The demo example can have title text that goes into `Title_Text` annotation, must have intro text annotated with `Intro_Text`, and optionally has references annotated with `Reference`. The example itself can be text (annotated with `Body_Text`) or it can be sample code. There may be more than one instance of example. All texts (`Body_Text`) are annotations of `Demonstrative_Example` individual.

`Example_Code` class and an object property with the same name associate the demo example with the example code. Individual IRI and a sequence number starting from zero form the demonstrative example IRI extended with the string "_EC". For example, CWE-609_Demonstrative_Example0 has two example code individuals CWE-609_Demonstrative_Example0_EC0 and CWE-609_Demonstrative_Example0_EC1.

`Structured_Code` annotation represents the example code itself, which is XHTML text. `Language` and `Nature` data properties represent the code attributes, which are enumerations.

`Language` data property is like `LanguageName`, but here it is used for consistency with the name formation for the other applicable platforms.

The observed examples should at best be CVE vulnerabilities, but this may not be the case. Therefore, `Observed_Example` class and an object property

with the same name associate the weakness observed examples.

`Observed_Example_Description` annotates the observed example description.

`Observed_Example_Reference` object property references CVE vulnerability as observed example. However, if there is no CVE vulnerability referenced, then the reference goes into `Reference` annotation, since `Link` links to an observed example anyway.

`Link` of the observed example is an URL, i.e. this is a location in Internet where to find the information about the observed example.

The functional areas indicate in which of them the weakness is most likely to manifest itself. `Functional_Area` data property represents it as enumeration.

`Affected_Resource` data property represents the affected resources in case of the weakness (vulnerability) exploitation. It is enumeration.

3.4 External references

External references are additional sources of information. They are descriptive in nature and their entire structure is embedded in the annotation `External_Reference`.

4 Conclusion

The CWE ontology presented here is a part of a cybersecurity ecosystem developed on the CVE/NVD, CWE, and CAPEC databases.

The process of knowledge formalization of CWE database has discovered several problems in the existing database. The solutions of these problems in the CWE ontology are directions for further CWE schema improvements. CWE database is evolving and it is normal to contain some contradictions.

Users can use directly CWE ontology if they have experience in SPARQL, RDF, OWL, reasoners etc. This is not realistic assumption about cybersecurity experts. Therefore, further research is on the development of specific interfaces to the ontology supporting the cybersecurity expert tasks.

For example, a typical task is a cyber forensic. An intrusion detection is a result of an attack. The last has exploited some vulnerabilities. Therefore, the expert knowing the attacked system has to find exploited vulnerabilities. CWE classifies the vulnerabilities.

This job the expert can do using the corresponding databases (CVE/NVD, CWE, and CAPEC) querying them applying the knowledge about the attacks,

vulnerabilities and weaknesses. However, reasoning the ontologies can automate this process.

The above-mentioned use case of CWE ontology is only one the many use cases of pre-exploit and post-exploit cybersecurity activities.

CWE ontology is one of the three components of the cybersecurity ecosystem. Its usability is in the ecosystem context. It is not possible to cover all topics, so only some common notes and considerations have shared here.

There are two main categories of cybersecurity activities: pre- and post-exploit activities.

The pre-exploit activities can use the ecosystem. These activities guard the system from known threads – attacks and vulnerabilities. Such information is available in the ecosystem.

For the post-exploit activities, the ecosystem is less usable especially if the attack or exploited vulnerability is unknown. In the last situation, weaknesses have to classify the vulnerability for the post-exploit activities. The reasoning in the CWE ontology can automate this classification.

A Python program generates CWE ontology. This generator address is available at https://github.com/VladimirDimitrov1957/CWE-ontology-generator.

Finally, several words about CWE ontology place in MIRACle project. Part of the project objectives is the development of autonomous guard systems of IoT type. These systems operate in naturally hostile environment. They use Wi-Fi connectivity. Therefore, security of these systems is in permanent issue resulting of the new security threads.

Above mentioned autonomous systems have to be developed and maintained with strong security support in the design. The cybersecurity ecosystem is the context to achieve it. CWE ontology is a part this ecosystem.

Acknowledgements

This paper is prepared with the support of MIRACle: Mechatronics, Innovation, Robotics, Automation, Clean technologies – Establishment and development of a Center for Competence in Mechatronics and Clean Technologies – Laboratory Intelligent Urban Environment, funded by the Operational Program Science and Education for smart growth 2014-2020, Project BG 05M2OP001-1.002-0011.

References

[1] MITRE Corporation, Common Weakness Enumeration (CWE), https://cwe.mitre.org/.

[2] MITRE Corporation, Common Vulnerabilities and Exposures (CVE), https://cve.mitre.org/.

[3] NIST, National Vulnerability Database (NVD), https://nvd.nist.gov/.

[4] NIST, Security Content Automation Protocol, https://csrc.nist.gov/projects/security-content-automation-protocol.

[5] MITRE Corporation, Common Attack Pattern Enumeration and Classification (CAPEC), https://capec.mitre.org/.

[6] V. Dimitrov, "IoT Security Issues", *Information Systems & Grid Technologies: Fifteenth International Conference ISGT'2022*, May 27–28, 2022, Sofia, Bulgaria, pp. 211-219, CEUR Workshop Proceedings (CEUR-WS.org), https://ceur-ws.org/Vol-3191/paper19.pdf