Annotation of CVE Descriptions

Vladimir Dimitrov

Faculty of Mathematics and Informatics
University of Sofia St. Kliment Ohridki, 5 James Bourchier Blvd., 1164, Sofia, Bulgaria

cht@fmi.uni-sofia.bg

Abstract. Knowledge extraction from texts is based on the text annotation. Text annotation process in essence understands of the text contents. This process intensively uses knowledge that cannot be found in the annotated text. The aim of this research is to a generate knowledge base from CVE descriptions.

Keywords: text annotation, knowledge base, ontology, CVE, vulnerability.

1 Introduction

The MITRE Corporation maintains a public database for weaknesses, namely CWE [1] and a public database for vulnerabilities, known as CVE [2].

The ontology must be simplified to be usable for educational purposes.

- "Weakness-a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. This term applies to mistakes regardless of whether they occur in implementation, design, or other phases of the SDLC."
- "Vulnerability-an occurrence of a weakness (or multiple weaknesses) within software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness."

The focus of this research is on the vulnerabilities, i.e. CVEs. Here, the weaknesses (CWEs) are vulnerability types.

2 CWEs and CVEs

The CWE database is organized in several views intended for different auditoria. A view may be structured by categories. The last ones are conceptual elements structuring the weaknesses. CWE views for researchers, for developers, and for architects are structured by categories. Each category can contain subcategories.

The *classes*, *bases* and *variants* are kind of weaknesses at different abstraction levels. The class is an abstract weakness that is not associated with any platform or technology. Bases are more specific than classes. The base usually is not associated with any platform or technology but contains enough details to be detected. The variant is more specific than the base and is usually associated with a specific platform or technology.

The weaknesses are organized in abstraction levels but not in inheritance hierarchies. A class can be more abstract than other classes, bases and variants. A base can be more abstract than other bases and variants. A variant can be more abstract than other variants.

Compound weaknesses (*composites* and *chains*) combine several other simple weaknesses. The chains are ordered while the composites are simply sets.

The weaknesses participate in more than one view, but it is possible for a weakness to participate within a view more than once.

Weaknesses are organized by structure and by abstraction, but there are relations among them.

CWE entry includes the following fields: CWE ID and name; description; alternate terms; description of the behavior; description of the exploit; likelihood of exploit; description of the consequences of the exploit; potential mitigations; node relationships; source taxonomies; code samples for the languages/ architectures; CVEs (vulnerabilities) for which that type of weakness exists; and references.

Weaknesses are vulnerability types. Each CWE references CVEs of its type. CVEs are classified by CWEs.

Initially, the vulnerability is registered as CVE, but usually, its type is not clear. After some investigations, a type (or types) is assigned to this new vulnerability. If there are no suitable CWEs, a new CWE can be created.

The investigation process can identify the conducted attack types for investigated CVE. Attack types are classified as templates in CAPEC (Common Attack Pattern Enumeration and Classification) [3] by MITRE Corporation.

Sometimes, it is impossible to identify the vulnerability type or its attack pattern. In these cases, corresponding references are not created in the CWE.

CWEs are the cornerstone for cybersecurity activities. They contain information for a vulnerability and possibly: how to identify, protect, detect, respond, and recur from it.

CVE database is very simple. Each CVE entry has a name, description, references to external sources, and some maintenance information.

NVD (National Vulnerability Database) [4] is based on CVE database. Each CVE entry in NVD contains some metrics.

The CVE description must follow one of the next two patterns as described in [2]:

- [VULNTYPE] in [COMPONENT] in [VENDOR] [PRODUCT] [VERSION] allows [ATTACKER] to [IMPACT] via [VECTOR].
- [COMPONENT] in [VENDOR] [PRODUCT] [VERSION] [ROOT CAUSE], which allows [ATTACKER] to [IMPACT] via [VECTOR].

The product ([PRODUCT]) can be identified in the next combinations:

- "[VENDOR_NAME] [PRODUCT_NAME]",
- "[PRODUCT_NAME]", with keywords (the product has no name),

- the product name is written as the vendor names it,
- "[PRODUCT_NAME] (aka [ALT_NAME])",
- "[PRODUCT_NAME] ([ACRONYM])",
- "[PRODUCT_NAME (formerly [OLD_NAME])",
- "[PRODUCT_NAME] and [OTHER_PRODUCT_NAME]",
- "[PRODUCT_NAME], as used in [BUNDLING_PRODUCT]",
- "[PRODUCT_NAME] [COMPONENT_TYPE] for [PLATFORM]".

The version ([VERSION]) can be represented in several variants:

- "The version 1.2.3"
- "The versions 1.2.3, 2.3.1, and 3.1.2",
- "The version 1.2.3 and earlier",
- "The versions 1.2.3, 2.3.1, 3.1.2, and earlier",
- "The versions before 1.2.3",
- "The versions before 1.2.3, 2.x before 2.3.1, and 3.x before 3.1.2",
- "The versions 1.2.1 through 1.2.3",
- "The versions 1.2.1 through 1.2.3 and 2.0.1 through 2.3.1",
- "The versions 1.2.3, 2.0.3 before 2.3.1, and 3.0.1 through 3.1.2",
- "Product A 1.2.3 and Product B 4.5.6",
- "Product A 1.2.3, 2.3.1, and 3.2.1 and Product B 4.5.6, 5.6.4, and 6.5.4".

When [VERSION] is used in disclosure phrasing, the combinations are:

- "Tested: 1.2.3",
- "Tested 1.2.3. Earlier versions are affected.",
- "Fixed in 1.2.3",
- "1.2.3 to 2.3.1 or Tested: 2.3.1. Introduced in 1.2.3",
- "1.2.3 and later",
- "Product A 1.2.3 and Product B 2.3.4",
- "v1.2.3".

The [ATTACKER] can be remote attackers, remote authenticated users, local users, physically proximate attackers, remote [TYPE] servers, guest OS users, guest OS administrators, context dependent attackers, attackers, [EXTENT] user assisted [ATTACKER], and man-in-the-middle attackers.

The [VULNTYPE] is descriptive, but it is possible for more than one vulnerability type (CWE) to be applicable or for more than one component to be affected. Pattern examples given in [2] are:

- Cross-site scripting (XSS) vulnerability in [COMPONENT] in [VENDOR] [PRODUCT] [VERSION] allows remote attackers to inject arbitrary web script or HTML via the [PARAM] parameter.
- Multiple cross-site scripting (XSS) vulnerabilities in [VENDOR] [PRODUCT] [VERSION] allow remote attackers to inject arbitrary web script or HTML via the [PARAM] parameter to (1) [COMPONENT1], (2) [COMPONENT2], ... or (n) [COMPONENTn].
- Multiple cross-site scripting (XSS) vulnerabilities in [COMPONENT] in

[VENDOR] [PRODUCT] [VERSION] allow remote attackers to inject arbitrary web script or HTML via the (1) [PARAM1], (2) [PARAM2], ..., or (n) [PARAMn] parameter.

- Multiple cross-site scripting (XSS) vulnerabilities in [VENDOR] [PRODUCT] [VERSION] allow remote attackers to inject arbitrary web script or HTML via the (1) [PARAM1] or (2) [PARAM2] parameter to [COMPONENT1]; the (3) [PARAM3] parameter to [COMPONENT2]; ...; or (n) [PARAMn] parameter to [COMPONENTm].
- SQL injection vulnerability in [COMPONENT] in [VENDOR] [PRODUCT] [VERSION] allows [ATTACKER] to execute arbitrary SQL commands via the [PARAM] parameter.
- Multiple SQL injection vulnerabilities in [VENDOR] [PRODUCT] [VERSION] allow [ATTACKER] to execute arbitrary SQL commands via the [PARAM] parameter to (1) [COMPONENT1], (2) [COMPONENT2], ..., or (n) [COMPONENTn].
- Multiple SQL injection vulnerabilities in [COMPONENT] in [VENDOR] [PRODUCT] [VERSION] allow [ATTACKER] to execute arbitrary SQL commands via the (1) [PARAM1], (2) [PARAM2], ..., or (n) [PARAMn] parameter.
- Multiple SQL injection vulnerabilities in [VENDOR] [PRODUCT] [VERSION] allow [ATTACKER] to execute arbitrary SQL commands via the (1) [PARAM1] or (2) [PARAM2] parameter to [COMPONENT1]; the (3) [PARAM3] parameter to [COMPONENT2]; ...; or (n) PARAMn] parameter to COMPONENTm].

The [VECTOR] is the input and/or processes required to exploit the vulnerability. It is possible several attack vectors to be applicable for the same vulnerability.

The [COMPONENT] is a product part. A component can be a trigger point where the error occurs (may be in multiple places) or interaction point that accepts the vectors.

It is possible for a component to be unknown – in that case, it is skipped in the phrasing.

In addition, the message payload can be used as a vector or as a component.

There are rules for combination of vectors and components as listed below:

- There are two possible component locations: after the vulnerability type, but before the product name; after the vector.
- Trigger point goes before the product name.
- Interaction point goes after the vector. Component goes before the product if you are unsure which type of component it is; you think the component can be both a trigger and an interaction point.
- For multiple component/vector pairs components always go after the vector, no matter their type; dot notation is used.

The aim of this research is to extract knowledge from CVEs descriptions. For that purpose, GATE [5] is used. In the next section, GATE is briefly described.

3 GATE Environment

Our intention is to annotate CVEs descriptions in a way that permit automatically to generate ontology individuals for each CVE.

What is GATE? GATE is an open source solution for all live cycle of text processing. There are many GATE modules, but here the focus is on the GATE Developer, which is an integrated environment for language processing development. Its purpose is information extraction from text annotations.

GATE has many components (language, processing, and visualization resources). The standard set of resources is called CREOLE (a Collection of Reusable Objects for Language Engineering) [6].

ANNIE (A Nearly-New Information Extraction system) [7] is a CREOLE subset of components tuned for English language. It intensively uses components implemented in JAPE (Java Annotation Patterns Engine) [8].

GATE is also a template work process for language engineering. ANNIE components' arrangement within the standard workflow is as follows:

1. GATE inputs a single document or a set of documents (corpora). All corpora documents must have the same format. Among accepted by GATE document formats are XML, HTML, SGML, plain text.
2. Initially, the document is tokenized in words, numbers, and punctuation. English Tokenizer or Unicode Tokenizer can be used. Tokens are annotations that have attributes.
3. Then, the tokenized text can be processed with POS Tagger that annotates parts of the speech, such as noun, verb, adjective, etc.
4. Gazetteer annotate the text with known names. Essentially, it uses preprepared lists of names.
5. Sentence Splitter annotates the sentences in the text using language punctuation rules.
6. Semantic Tagger annotates some well-defined kinds of text: Person, Location, Organization, Money, Percent, Date, Address, Identifier and Unknown.
7. OrtoMatcher does not introduce new named annotations, but assigns types to unclassified proper names.
8. Pronominal Coreference annotates quoted texts and process pronouns.

The user can modify GATE components, can create new components and can rearrange process components because GATE source is freely distributed.

Ontologies can be used with Onto Gazetteer for text annotation. The user can develop in JAPE components that fully manipulate ontologies and create new individuals within them.

4 Annotating CVEs

The first step is to load CVE documents into GATE. For this step, corpora have to be created.

CVE database is available as one XML document. Every Vulnerability element in it is a CVE. GATE's import process can be configured to separate each Vulnerability element as a different document in the corpora.

CVE database is available in two formats: the original CVE format and in CVRF. The last one is simpler and contains only the last updated version – it is more suitable to be imported in GATE.

CVEs are more than 128 000 and as result of that, the loading process is very slow. It is recommended to create five corpora and to load them with around 25 000 documents – GATE fails to import more than 30 000 documents.

Then English Tokenizer tokenizes the corpora documents. It is recommended to save the XML tags in the result annotation set.

The next processing steps follow the standard procedure: POS Tagger, Gazetteer, Sentence Splitter, Semantic Tagger, OrtoMatcher, and Pronominal Coreference.

The key problem in the CVE descriptions annotation are product and vendor combination. For example, it is possible the vendor name to be part of the product name. The product and the vendor have key positions in the phrasing template that facilitate the recognition of other phrasing elements. All product and vendor names are listed at [9]. These lists can be used with Gazetteer to annotate products and vendors.

The annotation of the other elements from the CVE phrasing template ([VULNTYPE], [COMPONENT], [VERSION], [ATTACKER], [IMPACT], [VECTOR], and [ROOT CAUSE]) requires the development of a processing component in JAPE.

[VULNTYPE] has to be a CWE, but usually vulnerability types in CVE description do not refer to a CWE. In the best case, a vulnerability type is a CWE name (without the enumeration). How to deal with this problem?

The first approach is to extract all CWE names into lists and to use Gazetteer to annotate vulnerability types.

On the other hand, the vulnerability type has a fixed position in both phrasing templates and this fact can be used to annotate them.

At this stage of the research, it is not clear which approach to be used for vulnerability type annotation. May be a combination of them is better. Anyway, some manual work must be done.

The [COMPONENT] has no keywords or cannot be extracted from some lists, but they have fixed positions in the phrasing templates. The key for their annotation is vendor and product annotation must precedes that annotation.

The same considerations are applicable to [VERSION], [IMPACT], [VECTOR], and [ROOT CAUSE].

The situation with [ATTACKER] is better, because there is some keyword phrasing for it.

5 Conclusion

Annotated CVE descriptions can be used to generate ontology individuals. A GATE processing component has been developed and tested successfully. The corresponding CVE ontology has been developed, but it description is out of the scope of this paper.

GATE annotation processing components for key elements in CVE phrasing template have been implemented, but their recognition efficiency is still not satisfactory. For that purpose, additional research on the real CVE descriptions will be done to increase the recognition power of the component. Unrecognized elements from this component must be annotated manually, which exists as an option in GATE Developer.

6 Acknowledgements

This research is supported by the National Scientific Program “Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICTinSES)”, financed by the Ministry of Education and Science.

References

1. MITRE Corporation, Common Weakness Enumeration (CWE), http://cwe.mitre.org, accessed 20.02.2020.
2. MITRE Corporation, Common Vulnerabilities and Exposures (CVE), http://cve.mitre.org, accessed 20.02.2020.
3. MITRE Corporation, Common Attack Pattern Enumeration and Classification (CAPEC), http://capec.mitre.org, accessed 20.02.2020.
4. NIST, National Vulnerability Database (NVD), http://nvd.nist.gov, accessed 20.02.2020.
5. GATE, http://gate.ac.uk, accessed 20.02.2020.
6. GATE, Chapter 4. CREOLE: the GATE Component Model, http://gate.ac.uk/sale/tao/splitch4.html, accessed 20.02.2020.
7. GATE, Chapter 6. ANNIE: a Nearly-New Information Extraction System, http://gate.ac.uk/sale/tao/splitch6.html#chap:annie, accessed 20.02.2020.
8. GATE, Chapter 8. JAPE: Regular Expressions over Annotations, http://gate.ac.uk/sale/tao/splitch8.html, accessed 20.02.2020.
9. CVE Details, The ultimate security vulnerability datasource, http://www.cvedetails.com, accessed 20.02.2020.