Semi-Automatic Annotation of Natural Language Vulnerability Reports

Yan Wu, College of Information Science and Technology, University of Nebraska at Omaha, Omaha, NE, USA

Robin Gandhi, College of Information Science and Technology, University of Nebraska at Omaha, Omaha, NE, USA

Harvey Siy, College of Information Science and Technology, University of Nebraska at Omaha, Omaha, NE, USA

ABSTRACT

Those who do not learn from past vulnerabilities are bound to repeat it. Consequently, there have been several research efforts to enumerate and categorize software weaknesses that lead to vulnerabilities. The Common Weakness Enumeration (CWE) is a community developed dictionary of software weakness types and their relationships, designed to consolidate these efforts. Yet, aggregating and classifying natural language vulnerability reports with respect to weakness standards is currently a painstaking manual effort. In this paper, the authors present a semi-automated process for annotating vulnerability information with semantic concepts that are traceable to CWE identifiers. The authors present an information-processing pipeline to parse natural language vulnerability reports. The resulting terms are used for learning the syntactic cues in these reports that are indicators for corresponding standard weakness definitions. Finally, the results of multiple machine learning algorithms are compared individually as well as collectively to semi-automatically annotate new vulnerability reports.

Keywords: Common Vulnerabilities and Exposures (CVE), Common Weakness Enumeration (CWE), Machine Learning, Security Vulnerability, Semi-Automatic Annotation, Text Mining

1. INTRODUCTION

Software design and development is an error-prone process, which inadvertently introduces a number of weaknesses that can later be exploited for malicious purposes. A weakness is the result of a flaw that creates pre-conditions necessary for the introduction of vulnerabilities within that software. Weaknesses in software present opportunities for malicious attacks that compromise the security policy expectations. Common sources of weaknesses are validation errors, domain errors, and serialization/aliasing errors (Landwehr et al., 1994). Vulnerabilities

DOI: 10.4018/jsse.2013070102
and weaknesses are related but different, i.e. only exploitable (or already exploited) weaknesses can be considered as vulnerabilities. Weaknesses might stay in software but never cause any problems if they remain un-exploitable by an attacker. In this work we focus on annotating reports that document exploitable and publicly known vulnerabilities with corresponding standard weak-ness definitions. Several vulnerability reports in widely used software packages are cross-referenced from the National Vulnerability Database (NVD) (NIST, 2012) that maintains the Common Vulnerabilities and Exposures (CVE) list (MITRE, 2011b).

Most software development projects have some processes, tools and techniques to document and track reported vulnerabilities. This information is recorded in existing project repositories such as change logs in version control systems, entries in bug tracking systems and communication threads in mailing lists. As these repositories were created for different purposes, it is not straightforward to extract useful vulnerability-related information. In large projects, these repositories store vast amounts of data, and as a result, the relevant information is buried in a mass of irrelevant data. Natural language text descriptions and discussions do not facilitate mechanisms to aggregate vulnerability artifacts from multiple sources or pinpoint the actual software fault and affected software elements.

Two key problems exist: information overload of vulnerability information in software repositories and, paradoxically, lack of techniques to systematically annotate and analyze this data. The large volume of data in software repositories and other project information sources make it difficult to locate the artifacts needed to identify, track and study previously recorded vulnerabilities. This condition is compounded by the fact that the complete record of information is scattered over several separate systems with different information schemas and natural language descriptions. Even if the information is found, a significant amount of work is needed to reconstruct the trail of artifacts that help understand the actual vulnerability. Thus, the information within software project repositories is not in a representation that can be easily extracted and analyzed for vulnerability-related questions.

On the other hand, while a significant body of knowledge exists for classifying and categorizing software weaknesses, it is hardly applied in the context of a software project. From the information assurance community, there have been several attempts to catalog the knowledge and information gathered from past vulnerabilities in order to avoid future occurrences. The Common Vulnerabilities and Exposures (CVE) (MITRE, 2011b) provides an identification scheme that enables the collection and recording of known vulnerabilities from software development organizations, coordination centers and individuals. This facilitates review by experts which in turn leads to the generalization of similar vulnerabilities into higher-order software weaknesses. Several such software weakness categorizations have been attempted. Among these, the Common Weakness Enumeration (CWE) (MITRE, 2011c) is a community developed dictionary of such software weakness types that attempts to encompass other weakness categorization efforts.

Ideally, vulnerability reports should be linked to corresponding CWE entries. In practice, few vulnerability reports are categorized into weaknesses. Furthermore, as CWE tries to be inclusive of other attempts at categorizing software weaknesses, the CWE has become large and abstract, contains multiple views, with overlapping descriptions. This presents a huge cognitive overload for understanding the precise conditions that determine the existence of a weakness, its preceding software faults, resources/locations that it occurs in, and following consequences. As shown in Figure 1, even the natural language descriptions contain tangled concerns representing different aspects of a single weakness. These concerns are also repeated in closely related CWE entries (examine the overlapping concerns between CWE-119 and CWE-120 in Figure 1).
Related Content

Reengineering the Portal to Texas HistorySM: A Case Study
[www.irma-international.org/chapter/reengineering-portal-texas-historysm/77756/](www.irma-international.org/chapter/reengineering-portal-texas-historysm/77756/)

Evaluating the Understandability of Android Applications
[www.irma-international.org/article/evaluating-the-understandability-of-android-applications/191208/](www.irma-international.org/article/evaluating-the-understandability-of-android-applications/191208/)

An Ontology Based Representation of Software Design Patterns
[www.irma-international.org/chapter/ontology-based-representation-software-design/29408/](www.irma-international.org/chapter/ontology-based-representation-software-design/29408/)

Engineering Reusable Learning Objects
[www.irma-international.org/chapter/engineering-reusable-learning-objects/29418/](www.irma-international.org/chapter/engineering-reusable-learning-objects/29418/)

ART-Improving Execution Time for Flash Applications
[www.irma-international.org/chapter/art-improving-execution-time-flash/71988/](www.irma-international.org/chapter/art-improving-execution-time-flash/71988/)