An Ontology-based Approach to Model Common Vulnerabilities and Exposures in Information Security

Minzhe Guo[1] Ju An Wang[2]

Abstract – Machine understandable security vulnerabilities are in need for security content automation [2]. Common Vulnerabilities and Exposures (CVE) is an industry standard of common names for publicly known information security vulnerabilities, and has been widely adopted by organizations to provide better coverage, easier interoperability, and enhanced security [1]. In this paper, we focus our research on the problem domain of software vulnerability and propose an ontology-based approach to model security vulnerabilities listed in NVD [2], providing machine understandable CVE vulnerability knowledge and reusable security vulnerabilities interoperability. We illustrate the major design ideas of our ontology and give examples to illustrate how the ontology can be populated with the knowledge from standards. In addition, we also give examples to demonstrate the benefit of using ontology to study the nature of vulnerabilities and the relationships between vulnerabilities and its related areas.

Keywords: Ontology, CVE, Software Vulnerability

1. INTRODUCTION

Machine understandable security vulnerabilities are in need for security content automation [2]. Common Vulnerabilities and Exposures (CVE) is an industry standard of common names for publicly known information security vulnerabilities, and has been widely adopted by organizations to provide better coverage, easier interoperability, and enhanced security [1]. CVE is also a standard of Security Content Automation Protocol (SCAP), a U.S. government multi-agency initiative that aims at automating compliance, managing vulnerabilities and performing security management [2]. One problem with CVE is its syntactic description format. Currently, machine only knows that a vulnerability has been identified, but it does not understand the semantics of the vulnerability, i.e. it does not understand what product would be affected by this vulnerability, or what kind of damage would this vulnerability generate. Without a machine understandable CVE, the aim of SCAP could be hard to achieve.

An Ontology is an explicit specification of conceptualization [3]. It describes concepts and relationships between concepts at semantic level and knowledge level in order to express knowledge in a universal and machine-understandable way. Therefore, knowledge presented by an ontology can be shared and reused unambiguously among different implementations and organizations. Currently, several ontologies have been proposed in the information security domain [4, 5, 6]. To the best of our knowledge, no ontology about SCAP standards has been developed yet.

In this paper, we present an ontology-based approach to model security vulnerabilities listed in CVE, providing machine understandable CVE vulnerability knowledge and reusable security vulnerabilities interoperability. We construct our ontology based on the efforts of CVE, CWE, CAPEC and CPE, organizing the knowledge from those efforts to form a heavy-weight ontology which include concepts, concept taxonomies, relationships, properties, axioms and constraints.

---

[1]Southern Polytechnic State University, 1100 S Marietta Pkwy, Marietta, GA 30060,mguo@spsu.edu

[2]Southern Polytechnic State University, 1100 S Marietta Pkwy, Marietta, GA 30060,jwang@spsu.edu

The main goals of our current research activities are to build an ontology to formally represent various software vulnerabilities; to add semantics to vulnerability representation in order to be understandable by the machine, to provide a framework that integrates knowledge of vulnerabilities; to be abundance in vulnerability instances; and to study the nature and relationship of software vulnerabilities and related concepts.

The rest of the paper is recognized as follows: In section 2 we discuss the related work, while in section 3 we present the construction of our vulnerability ontology; the implementation and application examples of our ontology is presented in Section 4, and finally, conclusions and further research directions are given in section 5.

2. RELATED WORK

A number of papers have been published in the area of semantic technology, ontology, and applying semantic technologies to information security. Some effort focused on building security ontology to model the security requirements or security policy. Lee et al. [7] argue that the nonfunctional nature of security requirements, which are usually identified in regulatory documents for certification and accreditation activities, imposes complex constraints on behavior of software systems and makes them hard to understand, predict and control. Thus they propose to build problem domain ontology from regulatory documents enforced by the DITSCAP – Department of Defense Information Technology Security Certification and Accreditation Process. A common language for extracting concepts from regulatory documents is presented, as well. Amaral et al. [8] tries to formalize the text-based information in the domain of Information Security, such as security policies defined by organizations. They propose techniques used to extract knowledge from natural language texts to form the ontology which consists of a vocabulary for the Information Security domain, logical forms corresponding to statements in the text and a set of axioms used for inference. A tool providing automatic support for the formalization process is also described in their paper.

There are quite a number of papers focusing on building generic security ontology to support the information system security management or risk analysis. Tsoumas et al. [9] argue that a structured approach might be employed into Information System (IS) security management so as to support the process leading from informal, high-level statements found in policy and risk analysis documents to deployable technical controls. Authors extend the DMTF Common Information Model (CIM) standard in order to use it as a container for IS security-related information, and then enrich the CIM extension with ontological semantics in order to support knowledge sharing and reuse defining a generic Security Ontology (SO). Furthermore, the necessary steps to establish the IS security management framework is discussed in the paper, as well. Mouratidis et al. [10] identify the need to extend the Tropos ontology [11] to consider security issues. The Tropos ontology is based on social hierarchies and adapts components of the i* framework [12]. Authors improve the social ontology created for i* framework with new security concepts: security constraints, secure entities and secure dependences between actors. Ekelhart et al [13] propose a security ontology framework which unifies existing approaches to support IT-Security risk analysis. The framework consists of four components: a security ontology based on the security and dependability taxonomy by Landwehr [14], the underlying risk analysis methodology, concepts of the (IT) infrastructure domain and a simulation enabling enterprises to analyze various policy scenarios. Their recent researches on applying the security ontology to risk assessment can be found at [10, 15].

It is interesting to notice that some research work on using ontology to model security attacks. Undercoffer et al. [16] state the benefit of transitioning from taxonomies to ontologies and propose an ontology to model computer attacks for sharing the knowledge in intrusion detection systems. Authors use DAML+OIL and DAMLJessKB to implement the ontology and present use case scenarios to illustrate the benefit of utilizing the ontology. Vorobiev et al. [17] analyze and classify the Web services security threats systematically in order to build a security attack ontology with the objective to allow various firewalls and intrusion detection systems to share a common understanding of the attack knowledge and to allow the reasoning services and automatic analyzing.

Finally, there are a few papers concentrate on adding security to semantic web research. Denker et al. [18] have created several ontologies for specifying security-related information in Web Services first using DAML+OIL [19] and later OWL [18]. They defined ontology with the goal to enable high-level markup of web resources, services, and agents and to provide a layer of abstraction on top of various web service security standards. Two sub-ontologies - “security mechanisms” that captures high-level security notations and credentials defining different

authentication methods make up the ontology. Kim et al. [20] focus their research on annotation of functional aspects of resources and propose to build an NRL security ontology to represent security statements like mechanisms, protocols, algorithms and credentials. The NRL security ontology employs an architecture that is easy to use and easy to extend. Bao et al. [21] investigate how to secure the sharing of ontologies between autonomous entities. They provide a framework for privacy-preserving reasoning in order to allow an agent to safely answer queries against its knowledge base using inferences based on both the hidden and visible part of the knowledge base, without revealing the hidden knowledge.

Our approach differentiates with the previous work in the following aspects: (1) Our ontology focuses on the problem domain of software vulnerability; (2) The construction of our ontology is based on the widely accepted standards like CVE, CPE, CWE and CAPEC, and our ontology can be abundance in instances and relationships; (3) Our ontology can be used to study the relationship between vulnerabilities.

3. ONTOLOGY MODELING

3.1 Common Vulnerability and Exposures

The MITRE Corporation put together a list, called CVE, of publicly known information security vulnerabilities and exposures. Each vulnerability on this list has a unique identifier enabling data exchange between security products and providing a baseline index point for evaluating coverage of tools and services [22]. Each CVE identifier includes appropriate references. CVE is now the industry standard for vulnerability and exposure names. A vulnerability in CVE could be represented using XML as shown in Figure 1:

```
<item type="CAN" name="CVE-2008-5070" seq="2008-5070">
       <status>Candidate</status>
       <phase date="20081114">Assigned</phase>
       <desc>
              SQL injection vulnerability in Pro Chat Rooms 3.0.3, when
              magic_quotes_gpc is disabled, allows remote attackers to
              execute arbitrary SQL commands via the gud parameter to
              (1) index.php and (2) admin.php.
       </desc>
       <refs>
              <ref source="MILW0RM"
              url="http://www.milw0rm.com/exploits/6612">6612</ref>
              <ref source="BID"
              url="http://www.securityfocus.com/bid/31463">31463</ref>
       </refs>
       <votes>
       </votes>
       <comments>
       </comments>
</item>
```

Figure 1 Representing a vulnerability with XML

All information about a software vulnerability is incorporated in the `<item>` element and the attribute "`name`" is used to provide a unique name of the vulnerability. Other than the name of the vulnerability, the most important part of a CVE item is the `<desc>` element which is used to provide a brief and formal description about the vulnerability. Generally, the description consists of concepts that are needed to provide a meaningful description of the vulnerability, such as the type of the vulnerability, the affected IT system, the attacker, the consequences, etc.

Through the analysis of CVE vulnerability expressions, we concluded the syntax of the `<desc>` element that CVE used to describe vulnerabilities could be expressed with a simple Extended BNF (EBNF) as follows:

```
CVE_VULNERABILITY ::=
( IT_SYSTEM VERSION+ )+
|“because of ” REASON*
|“when” CONDITION*
|“in” (COMPONENT|FILE|FUNCTION)*
“has” (specified|unspecified) VULNERABILITY
```

```
“allow” (specified|unspecified) ATTACKER+
“to cause” CONSEQUENCE+
|“via” ATTACK*
```

Thus the vulnerability in Figure 1 could be expressed as:

```
CVE-2008-5070.DESC ::=
Pro Chat Rooms 3.0.3
when magic_quotes_gpc is diabled
has the SQL Injection Vulnerability
allow remote attackers
to cause arbitrary SQL commands execution
via the gud parameter to (1)index.php and (2)admin.php
```

From the syntax and examples described above, we notice that the symbols such as “`IT_System`”, “`Version`”, etc, that appear in the syntax are the minimally necessary concepts to describe a vulnerability, therefore they will be included in our vulnerability ontology described in the next section.

3.2 An Ontology for CVE Vulnerabilities

As discussed previously with CVE examples, we captured important concepts for describing vulnerabilities in the context of software security. Symbols and expressions like “VULNERABILITY”, “ATTACKER”, “COUNTERMEASURE” etc. are essential to characterize vulnerabilities, therefore, they should be defined in our vulnerability ontology. Figure 2 shows the conceptual model of our vulnerability ontology. The detailed attributes of each concepts and sub-concepts are not depicted here for the clarity of the diagram.

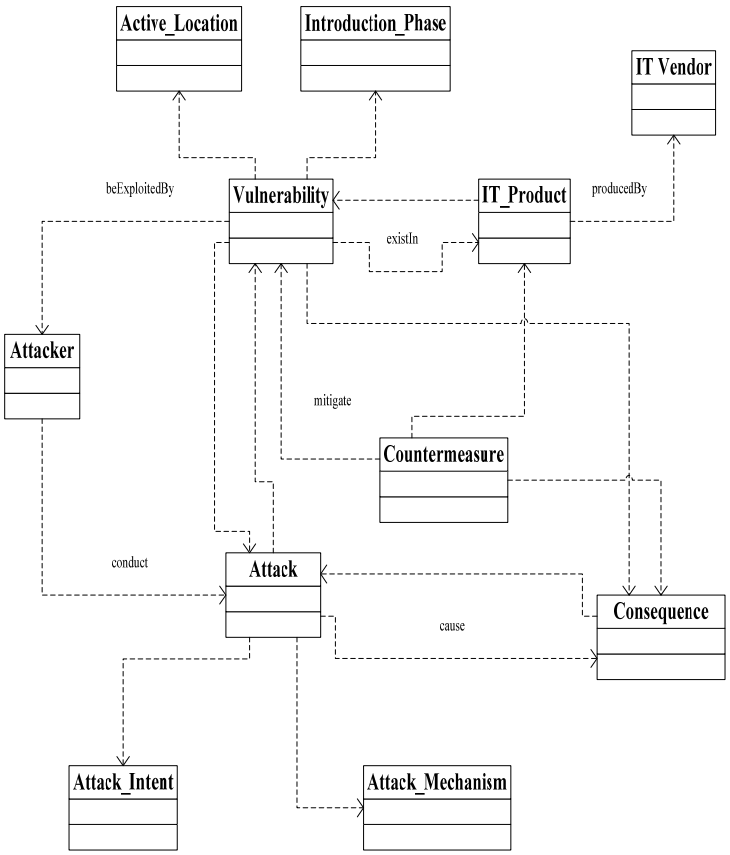

Figure 2 The Concept Model of the Vulnerability Ontology

The top level concepts of the ontology include: *Vulnerability*, *IT Product*, *Attacker*, *Attack*, *Consequence*, and *Countermeasure*. More specifically, a *Vulnerability* existing in an *IT_Product* can be exploited by an *Attacker* through conducting an *Attack* action with the objective to compromise the *IT_Product* and cause *Consequence*. *Countermeasures* can be used to protect the *IT_Product* through mitigating the *Vulnerability*. We provide more details about these top level concepts as follows.

(1) **Vulnerability:** *Vulnerability* refers to the flaws, defects, or mistakes in software that can be directly used by a hacker to gain access to a system or network. CVE considers a mistake a vulnerability if it allows an attacker to use it to violate a reasonable security policy for that system. This excludes those "open" security policies in which all users are trusted, or where there is no consideration of risk to the system. In order to better satisfy the properties of taxonomy [28] and our design objectives, we choose the CWE research view, a classification system in CWE, to be our vulnerability classification scheme. We take each weakness in CWE research view as a type of vulnerability, and thus a CVE vulnerability will become the instance of one or more types of vulnerabilities.

(2) **Introducation_Phase:** *Introduction_Phase* refers to the phases in the software development life cycle (SDLC), such as: requirements specification, design, coding, testing, integration, deployment, maintenance, etc, during which the vulnerability can be introduced. The taxonomy of this concept is based on the flaws by time of introduction from NRL taxonomy [30]. For instance, the vulnerabilities of the type “CWE-399: Resource Management Errors” can be introduced in the phase of implementation in the SDLC. With further details of this concept, the relationship between vulnerability and development lifecycle can be further investigated.

(3) **Active_Location:** *Active_Location* refers to the locations of the software system where the flaw manifests itself. For instance, one active location could be the system configuration files, where the vulnerability will be active during the system initialization. The taxonomy of this concept is based on the “flaws by location” from the NRL taxonomy. For example, the vulnerabilities of the type “CWE-399: Resource Management Errors” can be activated in the memory management module of the operating system.

(4) IT_Product: *IT_Product* is the concept that subsumes an enumeration of IT products encoded in CVE vulnerability descriptions. Each instance of *IT_Product* can be reflected to an external entity of IT system in the real word. We differentiate the software products from the hardware products, and divide the software products into two types – operating systems and applications.

(5) IT_Vendor *IT_Vendor* is the supplier of the *IT_Product*, who produces the instances of software or hardware products. The vendor can be a commercial IT company, an open source project, an academic institution, or an individual programmer. *IT_Product* together with *IT_Vendor* are the concepts that compose the targeted IT system that are described in the CVE vulnerability `<desc>` element. For example, the targeted IT system in CVE-2008-5044 is Microsoft Windows Server 2003 and Vista. We construct both concepts based on the Common Platform Enumeration (CPE) [29], which is a structured naming scheme for information technology systems, platforms, and packages, in order to provide an unambiguous naming for each targeted IT system. The CPE name structure is listing as follows:

```
cpe:/{part}:{vendor}:{product}:{version}:{update}:{edition}:{language}
```

The underlying idea is that best practices have greater utility when all participants share common names for the entities described and use of consistent and meaningful names can speed application development, foster interoperability, improve correlation of test results, and ease gathering of metrics. Figure 3 below uses the targeted IT systems in CVE-2008-5112 to present design of the structure of these two concepts.

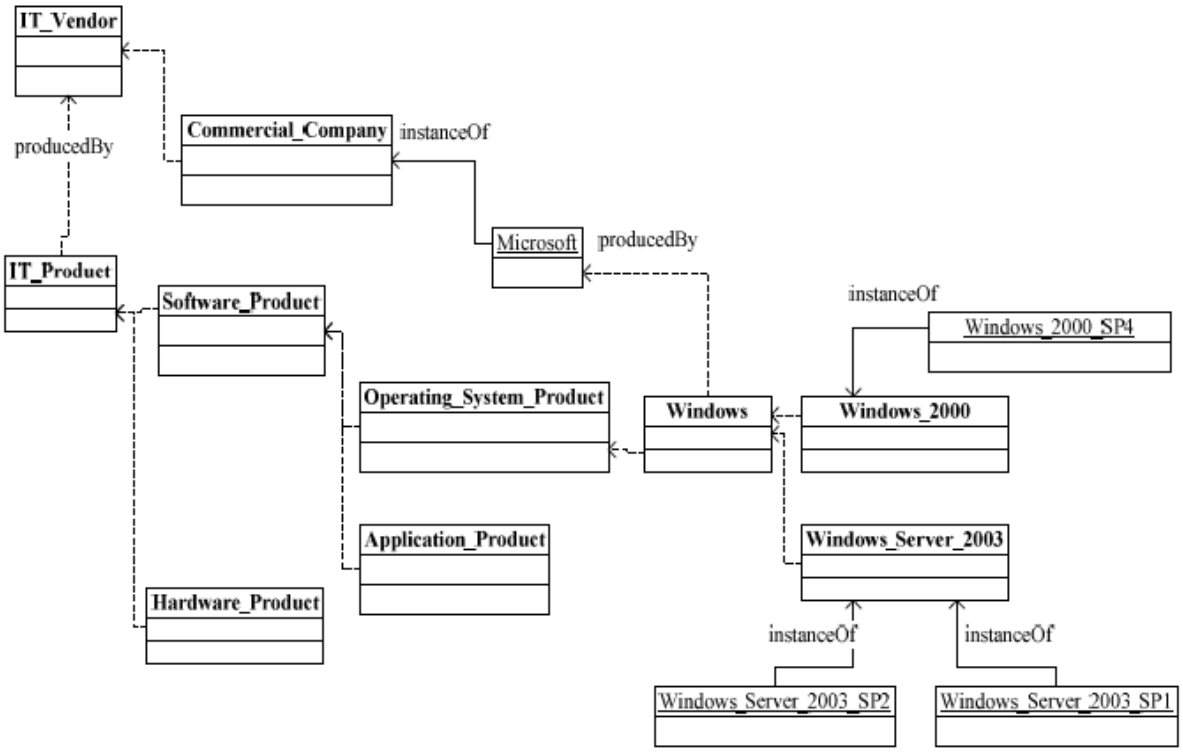

Figure 3 The targeted IT systems in CVE-2008-5112

Here Windows_Server_2003_SP1, Windows_Server_2003_SP2 and Windows_2000_SP4 in Figure 3 are all instances of Windows products, while Microsoft is an IT vendor which is an instance of Commercial_Company. The relationship "producedBy" is used to express the production relationship between an IT product and its vendor.

(6) **Attack**: *Attack* refers to a description of a common approach that can be taken by attackers to compromise the IT system through exploiting the vulnerabilities. In our ontology, we design the attack classification scheme based on the effort of Common Attack Pattern Enumeration and Classification (CAPEC) which divides the attacks into eleven categories including Abuse of Functionality, Spoofing, Exploitation of Authentication, Injection, etc. An attack pattern in CAPEC is identified by CAPEC ID. We take a single attack pattern as a type of attacks. For example, the vulnerability "CVE-2008-5070" is an instance of SQL Injection (CWE-89), and the attack that can exploit this vulnerability can be an instance of SQL Injection Attack (CAPEC-66).

(7) **Attack_Intent:** *Attack_Intent* is the concept to describe the generalized purpose of the attack. This class is comprised of several subclasses, each of which represents one type of purposes, including Exploitation, Penetration, and Obfuscation.

(8) **Attack_Method**: *Attack_Method* is the concept to describe the methods used in the attacks. This class is comprised of several subclasses, each of which represents one type of attack methods, including Injection, Spoofing, Protocol_Manipulation Analysis, Modification_of_Resources, API_Abuse, Social_Engineering, Time_and_State , Brute_Force.

(9) **Attacker**: *Attacker* is the concept to describe the human or agents that conduct the attack by exploiting the vulnerability to cause the consequences. This class is comprised of several subclasses, each of which represents one type of attackers, including Remote_Attacker, Local_Attacker. For example, CVE-2008-5070 can be exploited by an instance of Remote_Attacker.

(10) **Consequence:** *Consequence* refers to the technical part of impact that could be caused by the Attack conducted by Attackers through exploiting certain Vulnerabilities. This class is comprised of several subclasses, each of which represents one type of consequences, including Denial_of_Service, Privilege_Escalation, Data_Modification Information_Leakage, Arbitrary_Code _Execution. Take CVE-2008-5070 for example, the consequence that it might cause can be an instance of Arbitrary_SQL_Command_Execution, which is a subclass of Arbitrary_Code_Execution.

(11) **Countermeasure**: *Countermeasure* is the concept to describe actions or approaches that can potentially prevent or mitigate the risk of this attack. These solutions and mitigations are targeted to improve the resistance of the target software and thus reduce the likelihood of the attacker's success or to improve the resilience of the target software and thereby reduce the impact of the attack if it is successful. This concept is comprised of several sub concepts, each of which represents one type of countermeasures, including Patching and Advisory.

4. IMPLEMENTATION AND EXAMPLES

After designing the conceptual model, we implemented our ontology in OWL-DL [23] using Protégé [24], and instantiated it with the knowledge in CVE, CPE, CWE, CAPEC. Our implementation also employs the Pellet [26] as the reasoner for OWL-DL reasoning tasks and uses Jess [27] as the rule engine in order to allow writing Semantic Web Rule Language (SWRL) [25] to reason about OWL individuals and to infer new or implicit knowledge about those individuals.

From a logical point of view, our ontology is a knowledge representation system based on Description Logics, thus it is able to perform the basic reasoning tasks on both the TBox and ABox. The equivalence of OWL-DL and description logic allows OWL to exploit the existing body of DL reasoning to fulfill important logical requirements. These requirements include concept satisfiability, class subsumption, class consistency, and instance checking.

In addition, with the concepts, axioms and basic properties that have been designed in the ontology, it is now able to use it to help discover the complex relationship among individuals, among individuals and concepts and among concepts. Take the discovering of similarity relationship between two vulnerabilities for example, we may define the similarVulnerability relation as a binary relation between two vulnerabilities and it's an "owl:TransitiveProperty" relation. When two vulnerabilities can have the same genesis, they can exist in the same IT product, they can be exploited by the same kind of attackers and the same type of attack, causing the same type of consequence. Therefore, they will be recognized as an instance of similarVulnerability relation. The rule of similarVulnerability in SWRL would be as Figure 4:

```
Rule-similarVulnerability:
  Vulnerability(?x) ^ Vulnerability(?y) ^
  vulnerabilityName(?x, ?vn1) ^ vulnerabilityName(?y, ?vn2) ^  swrlb:notEqual(?vn1, ?vn2) ^
  tbox:isSubClassOf(?vc, Vulnerability)  ^
  abox:hasClass(?x, ?vc)  ^  abox:hasClass(?y, ?vc)  ^
  existInProduct(?x, ?p)  ^ existInProduct(?y, ?p)  ^
  IT_Product(?p)  ^ beExploitedBy(?x, ?z1)  ^
   beExploitedBy(?y,  ?z2)  ^  tbox:isSubClassOf(?zc,  Attacker)   ^   abox:hasClass(?z1,  ?zc)   ^
  abox:hasClass(?z2, ?zc) ^
  hasRelatedAttack(?x, ?a1)  ^  hasRelatedAttack(?y, ?a2)  ^
  tbox:isSubClassOf(?ac, Attack)  ^  abox:hasClass(?a1, ?ac)
  ^ abox:hasClass(?a2, ?ac)^hasRelatedConsequence(?x, ?c1)  ^ hasRelatedConsequence(?y, ?c2)  ^
  tbox:isSubClassOf(?c, Con)  ^
  abox:hasClass(?c1, ?c)  ^ abox:hasClass(?c2, ?c)
   → similarVulnerability(?x, ?y)
```

Figure 4. The rule of the similarVulnerability in SWRL

This rule is executed by Jess and the result of executing this rule would have the effect of adding the similarVulnerability property to each Vulnerability individual that satisfies our conditions. Also, we can use SQWRL [31] to query the ontology according to the rule above to help find the potential similar relationships between two Vulnerability instances. For example, we could query the vulnerabilities which are similar to the vulnerability "CVE-2008-0328" and the result shows that a vulnerability "CVE-2007-3652" can be a similar vulnerability of CVE-2008-0328. The query in SQWRL, the query result, and the description of two vulnerabilities in CVE are shown in Figure 5.

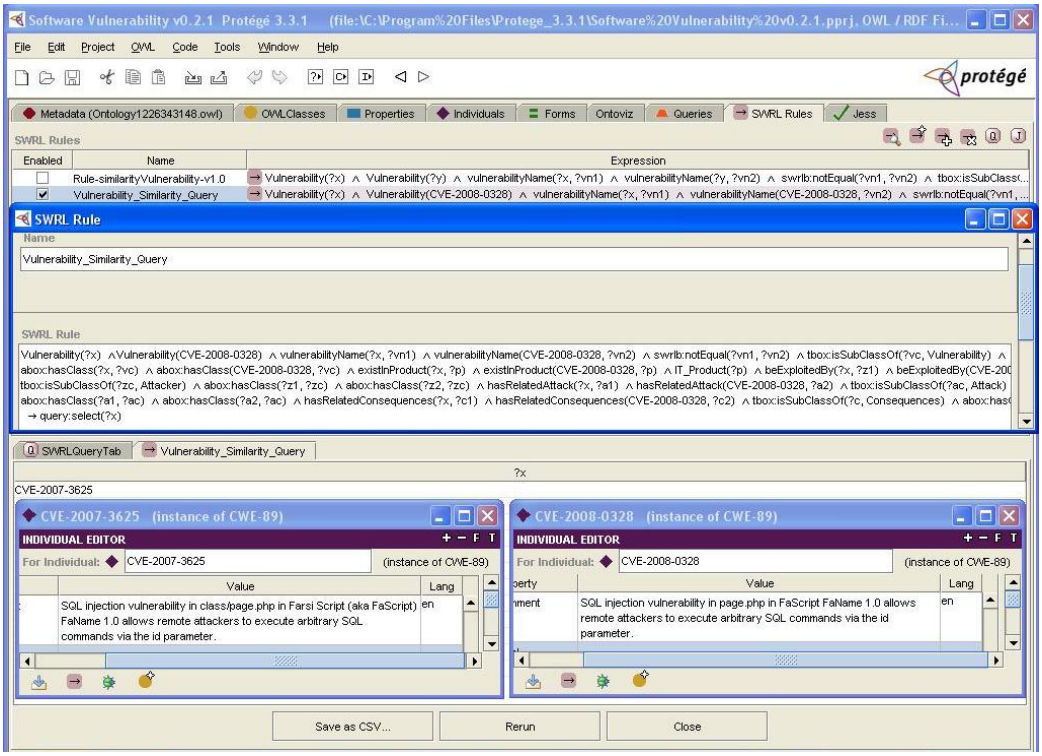

Figure 5. The result of the similarVulnerability Query

5. CONCLUSION

In this paper, we have studied the benefit of applying semantic technology to information security, in that the knowledge in information security areas can be formalized and shared by the community and can be automatically processed by computers. This work will certainly advance the automation of security content management and increase the effectiveness of security mechanisms.

We focused our research on the problem domain of software vulnerability and constructed our ontology based on those widely accepted standards like CVE, CPE, CWE and CAPEC. We illustrated the major design ideas of our ontology in this paper and examples were given as well in this paper to illustrate how the ontology can be populated with the knowledge from information security standards. In addition, we provided the example of similarVulneratility to demonstrate the benefit of using ontology to study the nature of vulnerabilities and the relationships between vulnerabilities and their related entities.

Currently we are continuing our research, refining the design and implementation of the ontology, and populating the ontology with more knowledge and instances in order to specify our ontology knowledge at a more comprehensive level and use it to study the relationships between vulnerabilities and other related security concepts.

For immediate future work, we will incorporate the knowledge from software development lifecycle, security policy and requirements, and risk management into our ontology, possibly through the integration and cooperation with other security ontology research in these areas, such that we can use the ontology to support high level security risk analysis, security metrics, and other stages in the lifecycle of security administration.

REFERENCES

[1] About CVE. [Online]. Available: http://cve.mitre.org/about [Sep, 2008].

[2] The Information Security Automation Program and the Security Content Automation Protocol. [Online]. Available: http://nvd.nist.gov/scap.cfm [Sep, 2008].

[3] Gruber T., "Towards Principles for the Design of Ontologies used for Knowledge Sharing". *International Journal of Human-Computer Studies*, 1995, 43(5/6): p.907-928.

[4] Denker, G. et al, "Security in the Semantic Web using OWL". *Information Security Technical Report*, 2005, 10(1): p.51-58.

[5] Tsoumas, B. et al, "Towards an ontology-based security management". *Proceedings of the 20th International Conference on Advanced Information Networking and Application (AINA '06)*, 2006.

[6] Ekelhart, A. et al, "Security Ontologies: Improving Quantitative Risk Analysis". *Proceedings of the 40th Hawaii International Conference on System Sciences (HICSS'07)*, 2007.

[7] Sea-Won Lee et al. Building Problem Domain Ontology from Security Requirements in Regulatory Documents. *In Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems*, 2006. ACM Press: Shanghai, China.

[8] Fernando Naufel do Amaral et al., An Ontology-based Approach to the Formalization of Information Security Policies. *Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops EDOCW '06*. IEEE Computer Society, 2006.

[9] Tsoumas, B. and D. Gritzalis, Towards an Ontologybased Security Management. *Proceedings of the 20th International Conference on Advanced Information Networking and Applications*. IEEE Computer Society, 2006. Volume 1 (AINA'06) - Volume 01 AINA '06.

[10] Mouratidis, H., P. Giorgini, and G. Manson, An Ontology for Modelling Security: The Tropos Approach. *Knowledge-Based Intelligent Information and Engineering Systems*, 2003, Springer Berlin / Heidelberg. p. 1387-1394.

[11] A. Fuxman, P. Giorgini, M. Kolp, J. Mylopoulos. Information Systems as Social Structures. *In Proceedings of the Second International Conference on Formal Ontologies for Information Systems (FOIS-2001)*, USA, 2001.

[12] E. Yu. Modeling Strategies Relationships for Process Reengineering. PhD Thesis, Department of Computer Science, University of Torento, Canada, 1995.

[13] Andreas Ekelhart, Stefan Fenz, Markus Klemen et al. Security Ontologies: Improving Quantitative Risk Analysis. *Proceedings of the 40th Annual Hawaii International Conference on System Science (HICSS'07)*, 2007.

[14] A. Avizienis, J. –C. Laprie, B. Randell, and C. E. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Sec. Comput.* Vol. 1, No. !. pp. 11-33, 2004.

[15] Formal Threat Description for Enhancing Governmental Risk Analysis.

[16] J. Undercoffer, A. Joshi and J. Pinkston. Modeling Computer Attacks: An ontology for Intrusion Detection. *In the sixth International Symposium on Recent Advances in Intrusion Detection*, 2003.

[17] A. Vorobiev and J. Han. Security Attack Ontology for Web Services. *Proceedings of the Second International Conference on Semantics, Knowledge, and Grid SKG '06*. IEEE Computer Society, 2006: p.42.

[18] G. Denker, L. Kagal, and T. Finin. Security in the Semantic Web using OWL. Security in the Semantic Web using OWL. *Information Security Technical Report*, 2005. 10(1): p. 51-58.

[19] Denker, G., et al., Security for DAML Web Services: Annotation and Matchmaking. *ISWC 2003*, 2003. Springer Berlin / Heidelberg. p. 335-350.

[20] Kim A., Security Ontology for Annotating Resources. *In 4th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE'05)*. 2005.

[21] Jie Bao, Giora Slutzki and Vasant Honavar. Privacy-Preserving Reasoning on the Semantic Web.

[22] Common Vulnerabilities and Exposures. [Online]. The MITRE Corporation. Available: http://cve.mitre.org/.

[23] Web-Ontology (WebOnto) working Group. http://www.w3.org/2001/sw/WebOnt/, November, 2008.
[24] Protégé. http://protege.stanford.edu/, November, 2008.
[25] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/, November, 2008.
[26] Pellet: The Open Source OWL DL Reasoner. http://clarkparsia.com/pellet/.
[27] Jess, the rule engine for the java platform. http://herzberg.ca.sandia.gov/.
[28] E. G. Amoroso. Fundamentals of Computer Security Technology. Prentice-Hall PTR, 1994.
[29] Common Platform Enumeration (CPE). http://cpe.mitre.org/, November, 2008.
[30] C. Landwehr, A. Bull, J. McDermott, and W. Choi , "A Taxonomy of Computer Program Security Flaws," *Computing Surveys*, 26 (3), pp. 211254 (Sep. 1994).
[31] SQWRL: Semantic Query-Enhanced Web Rule Language. http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL.

Minzhe Guo

Graduate Student of Information Technology Department, School of Computing and Software Engineering, Southern Polytechnic State University, mguo@spsu.edu.

Ju An Wang

Professor of Information Technology Department, School of Computing and Software Engineering, Southern Polytechnic State University, jwang@spsu.edu