VULCAN: Vulnerability Assessment Framework for Cloud Computing

Patrick Kamongi
Computer Science and Engineering
University of North Texas
Denton, TX 76203, USA
pkamongi@gmail.com

Srujan Kotikela
Computer Science and Engineering
University of North Texas
Denton, TX 76203, USA
ksrujandas@gmail.com

Krishna Kavi
Computer Science and Engineering
University of North Texas
Denton, TX 76203, USA
kavi@cse.unt.edu

Mahadevan Gomathisankaran
Computer Scienc and Engineering
University of North Texas
Denton, TX 76203, USA
mgomathi@unt.edu

Anoop Singhal
Computer Security Division
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
anoop.singhal@nist.gov

***Abstract*—Assessing security of software services on Cloud is complex because the security depends on the vulnerability of infrastructure, platform and the software services. In many systems, the platform or the infrastructure on which the software will actually run may not be known or guaranteed. This implies that the security of the software service must be assured regardless of the underlying infrastructure or platform, requiring a large number of combinations. Another common trend in Cloud and Service oriented Architecture (SoA) environments is Service composition, whereby new services can be created rapidly by composing existing services. Once again, the component services must be tested for security levels on a large number of platform and infrastructure combinations. In this paper we propose a novel vulnerability assessment framework for cloud computing systems. We have designed and developed a prototype of our framework.**

I. INTRODUCTION

Assessing security of software services on Cloud is complex because the security depends on the vulnerability of infrastructure, platform and the software services. The recent distribute denial of service cyber-attacks on American Banks websites [1] clearly shows the importance and imminence of cloud vulnerability assessment. It was discovered that various cloud services and public Web hosting services had been infected with a form of malware that has existed for years. A cloud based assessment framework could be used to discover this type of vulnerabilities and help to protect banks from being victims of known security vulnerabilities. In many systems, the platform or the infrastructure on which the software will actually run may not be known or guaranteed. This implies that the security of the software service must be assured regardless of the underlying infrastructure or platform, requiring a large number of combinations. Another common trend in Cloud and Service oriented Architecture (SoA) environments is Service composition, whereby new services can be created rapidly by composing existing services. Once again, the component services must be tested for security levels on a large number of platform and infrastructure combinations.

Vulnerability Assessment Framework is a structure supporting a set of tools that allows security practitioners to create and deploy exploits to find vulnerabilities. For example, Mercury [2] is one such assessment framework for Android based systems. In this work, we propose a novel vulnerability assessment framework for cloud computing systems. For the cloud systems, we can answer questions such as "I developed this cloud product as a service, is it vulnerable?". Or "I want to host this software application in this cloud environment, what security vulnerabilities I should watch out for?". Our framework provides a friendly interface to the user to learn more and assess their security in the cloud. In this paper, we present VULCAN a Vulnerability Assessment Framework for Cloud Computing. It provides a structure made of independent components and modules, whereas their combinatorial use allow us to assess security vulnerabilities for a given system.

Ontological Vulnerability Assessment [3] is an essential component of our framework. Using our Ontology Vulnerability Database (OVDB) [4] we provide two vital features to our framework. First feature, is the access to a conceptualized set of current known vulnerabilities listed in the National Vulnerabilities Database (NVD) [5]. The next feature, is using powerful ontology reasoning capabilities to search our knowledge base of vulnerabilities. And also, the ability to discover new vulnerabilities from the known existing one for a particular target system.

Automation is a vital aid in our framework. To use most updated information on the current known vulnerabilities, we automate the process of discovering, extracting them and populating our OVDB. Our vulnerability data sources comes from different repositories and sources such as NVD and web searches. Our framework allow us to do penetration testing as well. We use an approach of mapping our OVDB with attack exploits database such as Metasploit Auxiliary Module and Exploit Database [6]. Within the framework, both vulnerabilities and their exploits are mapped together, this provides a complete penetration testing environment.

The main purpose of our assessment framework VULCAN is to provide complete security vulnerability assessment for the cloud environment. To achieve this goal, we have proposed components and modules such as: Ontology Knowledge Base, Semantic Natural Language Processor, Indexer, and Vulnerability Class Index. We have designed each one of them to provide unique features. Our implementation of these features run on an environment that support Linux Operating System,

and Python. We used python language for developing our components and modules due to its flexibility and capabilities of powerful server-side scripting. Every component developed in our VULCAN framework can be integrated with any assessment framework, for example Metasploit [6], that support server side scripting. Finally our VULCAN framework is flexible that it allows integration of any additional component that could contribute to security vulnerability analysis.

Previous research on vulnerability assessment has yielded some solutions such as: the development of penetration testing tools, taxonomies and ontology of vulnerabilities, and assessment frameworks that allows integration of other components. However, few solutions for assessment in the cloud computing environment have been initiated. Our contribution in this area have focused on the cloud solutions such as:

1) The design and development of an assessment framework for the cloud environment.
2) Extended our previous ontology [4] definition for the cloud computing.
3) Designed an automated process for the ontology knowledge base creation from NVD data sources.
4) Proposed and designed necessary components and modules for vulnerability classification, and reasoning tasks for the cloud.

Within our VULCAN framework, we achieve:

1) Software vulnerabilities modeling
2) Analysis of vulnerabilities for cloud computing and mobile environments
3) Software penetration tool environment
4) Discovery of new vulnerabilities from the known one via the use of reasoning tasks on our ontology knowledge base.

The rest of this paper is organized as follows: In Section II we discuss the related work, while in Section III we present our framework architecture; the work flow of our assessment framework is explained in Section IV, the VULCAN implementation is detailed in Section V, and finally, conclusions and further research directions are given in Section VI.

II. RELATED WORK

Steele's [3] work on ontological vulnerability assessment shows that taking an ontological approach results in improved identification of complex vulnerabilities. In our current work, we see those results when using Protege (an ontology development environment) for generation and instantiation of our vulnerability ontology with NVD [5] data feeds. We are able to reason and query our ontology to find known vulnerabilities and discover unknown ones for a given target system.

Guo et al. [7] work present an ontology-based approach to model security vulnerabilities listed in Common Vulnerabilities and Exposures (CVE), providing machine understandable CVE vulnerability knowledge and reusable security vulnerabilities interoperability. Their efforts to form a well structured ontology which include concepts, concept taxonomies, relationships, properties, axioms and constraints, allowed us to extend their work into our OVDB.

The ontology for Vulnerability Management (OVM) [8] captures important concepts and relations for describing vulnerabilities in the context of software and system security. Their implementation of ontology in OWL-DL uses Protege 4, this task can become time consuming when looking at manually instantiating the ontology from a big data source like NVD. To overcome this challenge, we attempt to generate our ontology automatically using custom python scripts to extract relevant data for our ontology from NVD and generating our knowledge base (OVDB). Paul et al. [9] recommend the use of ontology to capture evolving requirements like in high assurance systems. Our OVDB allows us to capture anomalies and find vulnerabilities in the cloud systems.

Wang et al. [10] proposed an ontology–based approach to analyze and assess the security posture for software products. Normally, given a knowledge base of security vulnerability, you could retrieve currently known vulnerabilities of given target. Our attempt is on reasoning within ontology and to be able the discover new vulnerabilities that could exist in other products. Which will be caused by the presence of a vulnerability in one product that shares some underlying weak features with other products.

Xiao et al. [11] proposed a solution to overcome the tedious manual work on extracting Access Control Policies (ACP) from Natural Language (NL) documents. They proposed a solution 'Text2Policy', to automatically extract ACPs from NL software documents. This work relates to our attempt to automate our ontology generation from NVD data sources. We are using natural language processing techniques to extract data using pattern matching approach. We use the extracted data to populate our ontology.

The work in [12] presents a new method for automatic generation of OWL ontology from eXtensible Markup Language (XML) data sources. The proposed generation process is based on XML schema to build the ontology. However, this approach does not allow us to properly instantiate our ontology but still provides us a way to generate a useful ontology that we could produce from NVD data source.

The state of the art automatic ontology generation [13] defines its life cycle as a process composed of Extraction [acquisition of information needed to generate the ontology], Analysis [focuses on the matching of retrieved information and/or alignment of two or more existing ontology, depending on the use case], Generation [Ontology generation], Validation [Authenticate whether the generated ontology is correct or not], and Evolution [adapt to the ontology changes]. Our attempt is to come up with techniques to implement various components of the system for automatic generation of ontology and population of it from multiple corpuses of vulnerabilities.

In Meunier's [14] work, their contribution is a survey of currently known attempts to classify vulnerabilities and attacks. They illustrate how the current classifications fail to come up with one unified classification schema of all vulnerabilities and attacks. A recommended approach is to use ontology for vulnerabilities conceptualization. Because it is capable of adopting all kinds of vulnerabilities regardless of which sub categories they belong too. Our Ontology proves that the

recommended approach to be essentials when developing a vulnerability analysis assessment framework.

In the attempt to first create an assessment framework for Android, Mercury [2] was developed. Mercury is a framework that provides interactive tool that allows for dynamic interactions with the target applications running on a device. With Mercury it is possible to realize some of the attacks illustrated [15] against android security and can be learned via the proposed taxonomy of attacks on the Android operating system (OS). With our current ontology we could plug it into Mercury to assess security for android.

Attack graphs depict ways in which an adversary exploits system vulnerabilities to achieve a desired state [16]. In this work, they proposed a tool useful for generating and analyzing attack graphs. Our attempted work, is on using our ontology as a root node to discover known vulnerabilities of the target system, then initialize the attack graph generation for it.

In Heberlein et al. [17] work on establishing a taxonomic foundation for comparing and contrasting attack-graph approaches. We developed a similar approach on conceptualizing security vulnerabilities in our ontology.

III. Architecture

VULCAN uses ontology for creating vulnerability database and associates a vulnerability with one or more attack code snippets from the attack database. For assessing vulnerabilities in a specific domain, like cloud computing or mobile, we have organized our vulnerabilities into classes. In the following subsections the various components of VULCAN are described in detail.

A. NVD

National Vulnerability Database (NVD) [5] is a SCAP [18] compliant vulnerability database. The NVD database collects vulnerability information from various interrelated vulnerability databases like CVE [19], CWE [20], CPE [21], CVSS [22] etc. and compiles the information into a single database. Every entry in the NVD database is identified by a unique identifier. This identifier is referred to as CVE ID, which is an unique identifier for each vulnerability in the CVE database. This is the same identifier used across various other vulnerability databases mentioned above. A typical vulnerability entry in the NVD database has the vulnerability identifier, description of the vulnerability, list of software and their versions in which this vulnerability is found in, vulnerability severity score (CVSS) etc. collected from appropriate vulnerability databases. These vulnerability databases are industry standard databases maintained by MITRE. All the vulnerability information found in these databases is contributed by volunteers across the industry. The SCAP compliance of the NVD database makes it easy to inter-operate with other security tools and automate security assessment. VULCAN uses NVD as the source to populate vulnerability information into the ontology knowledge base.

B. OKB

Ontology Knowledge Base is the ontological database of vulnerability information from the NVD database. NVD provides the vulnerability database in a XML feed. We extract the vulnerability information from the XML feed and populate ontology knowledge base. The vulnerability information in the NVD XML feed is present in various tags. All the information in these tags are mapped to various classes and properties defined in the ontology.

C. System Classifiers

System Classifiers are dynamic inputs provided to the Indexer which will classify the classes in the ontology knowledge base. An example classification includes various vendors in the cloud computing domain and various software or hardware components in each service level of cloud computing services. As shown in Fig. 1, cloud computing domain is classified into IaaS, PaaS, SaaS etc. sub domains. In each of these domains we will include software and hardware components used in popular cloud computing vendors like Xen hypervisor in IaaS sub-domain, Google App Engine in PaaS sub-domain and Salesforce CRM in SaaS sub-domain. We can provide the system classifiers to whatever detail and depth we want to. The indexer takes these system classifiers as input and crawls through the ontology knowledge base and creates an index. The index consists of vulnerabilities grouped according the system classifiers provided by us. The changes in software or hardware in any domain or vendor would require updating the system classifiers and re-indexing the ontology knowledge base.

D. Indexer

Indexer is the software responsible for crawling through the ontology knowledge base an create an index. This index will in turn be used by the SNLP module to search the ontology knowledge base depending on the user query. The indexer is set to run every time the ontology knowledge base and/or system classifiers change. The indexer identifies all the vulnerabilities that are related to software or hardware components listed in the system classifiers and group them accordingly in the index.

E. Vulnerability Class Index

Vulnerability Class Index is the list of all vulnerabilities grouped into the categories provided by the system classifiers. These groups are called as ”Vulnerability Classes”. Vulnerability classes will assist users to search for vulnerabilities within a specific domain or sub-domain An example index looks like shown in the Fig. 1. At the top level there is cloud computing class. Cloud computing has a sub class called PaaS and the PaaS class has Xen hypervisor as it’s sub class. In the Xen class we have list of vulnerabilities extracted by the indexer from the ontology knowledge base.

F. SNLP

Semantic Natural Language Processor enables users to search and reason about vulnerabilities. It includes various sub components which are capable of doing pattern matching, keyword search, and reason over properties and relationships of the classes in the ontology knowledge base. SNLP takes input from user and tries to understand what the user is asking for and provides him a list of vulnerabilities for the requested product and/or class. SNLP is capable of looking up vulnerabilities for the requested product and listing vulnerabilities in a particular

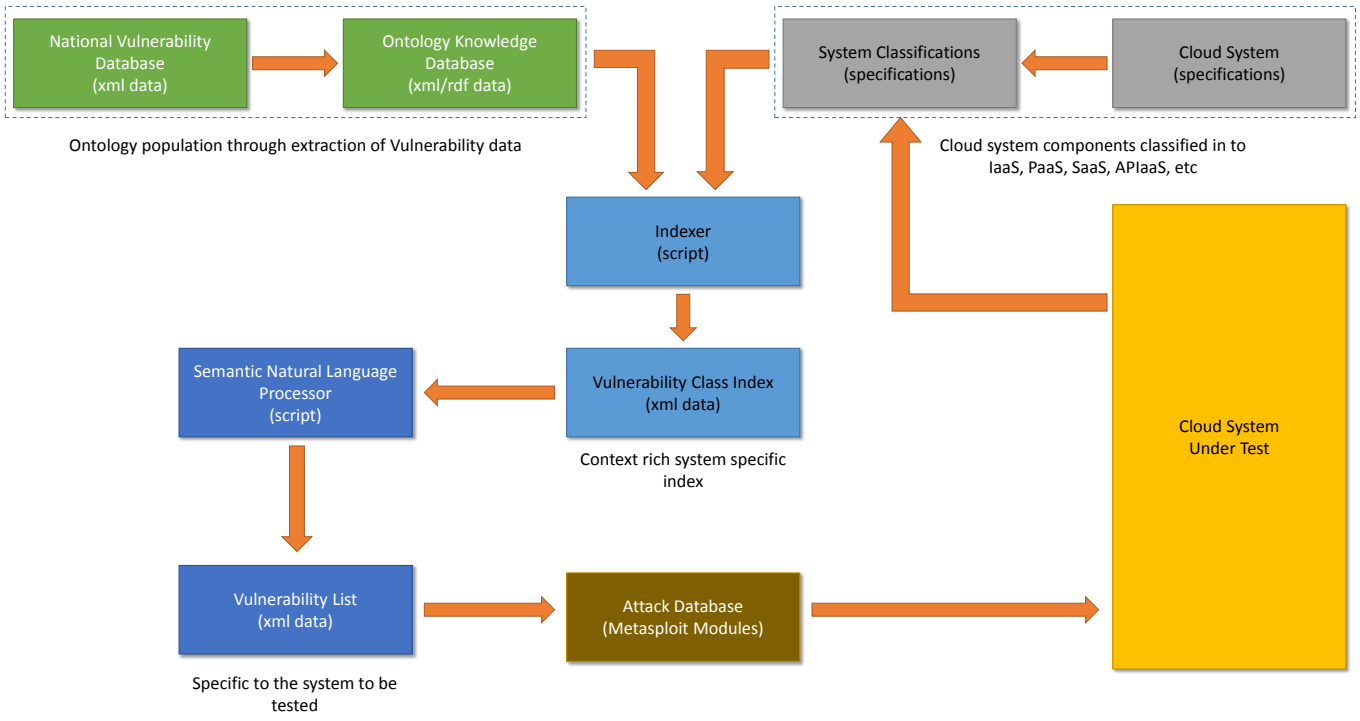

Fig. 1. VULCAN Architecture

class or product across various vendors. It also can reason and list vulnerabilities for the technology or framework used in the user's application.

IV. WORKING PROCESS FLOW

The NVD database consists of vulnerabilities identified by CVE ID and available as a XML data-feed from NIST. The instances for each vulnerability will be populated using XML parsing techniques. The entire XML data-feed is transformed into Ontology Knowledge Base (OKB). The OKB has classes, properties for vulnerabilities and relationships between these classes. This knowledge base with classes (and respective instances), relationships and properties will enable us to perform semantic queries and reason about vulnerabilities.

After populating the OKB, we provide a dynamic set of classifiers to the indexer. These classifiers are used to classify the vulnerabilities in the OKB. The indexer groups various vulnerabilities into classes of a specific sub-domain viz cloud computing, mobile computing etc. These classes help us to assess vulnerabilities of any application belonging to one of these sub-domains. Theses classifiers can be modified when any software or hardware component is modified in a particular sub-domain . For example, we may classify Xen vulnerabilities in Cloud Computing $\rightarrow$ Amazon $\rightarrow$ IAAS $\rightarrow$ Hypervisor class as Amazon uses Xen as the hypervisor. If in later point of time, Amazon decides to use KVM as hypervisor, we can update the classifiers accordingly. More details will be provided in the implementation section.

Once these classifiers are provided, the indexer creates an index with classes for the OKB. This index is referred by the SNLP module when a user performs a query. All the vulnerabilities matching user's application, technology and/or platform will be listed. User can then choose what vulnerabilities (s)he wants to test. Once the user selects the vulnerabilities he wants to test, necessary attacks are launched with the code from attack database. The wrapper scripts for the attack codes will provide necessary meta-data such as application path, necessary parameters. These attack scripts will launch attacks on the application and test it for the chosen vulnerabilities.

All the vulnerabilities tested positive will be reported to the user along with a security score based on the CVSS score. Necessary countermeasures will be provided if available. An illustration of our working framework is shown in Fig. 2. The implementation details for each component are detailed in the following section.

A typical use case scenario of using VULCAN components and modules to assess vulnerabilities for an android device using Mercury Framework [2] goes like this:

1) A User provides both dynamic inputs for example "Android" (this data is provided to the System Classifiers module of our VULCAN framework), and a natural language query for example "Assess for weaknesses that could allow an unauthorized access to my device?" (this query is processed within our VULCAN Semantic Natural Language Processor - 'SNLP').

2) The System Classifiers generates possible android based solutions and feeds them to the Indexer module. Then, the Indexer creates relevant vulnerabilities indexes which are used to produce vulnerabilities groups from the Vulnerability Class Index module. A sample created vulnerabilities group named "Root Access" contains indexed data of these CVE-IDs: CVE-2011-3874, CVE-2011-1823 and

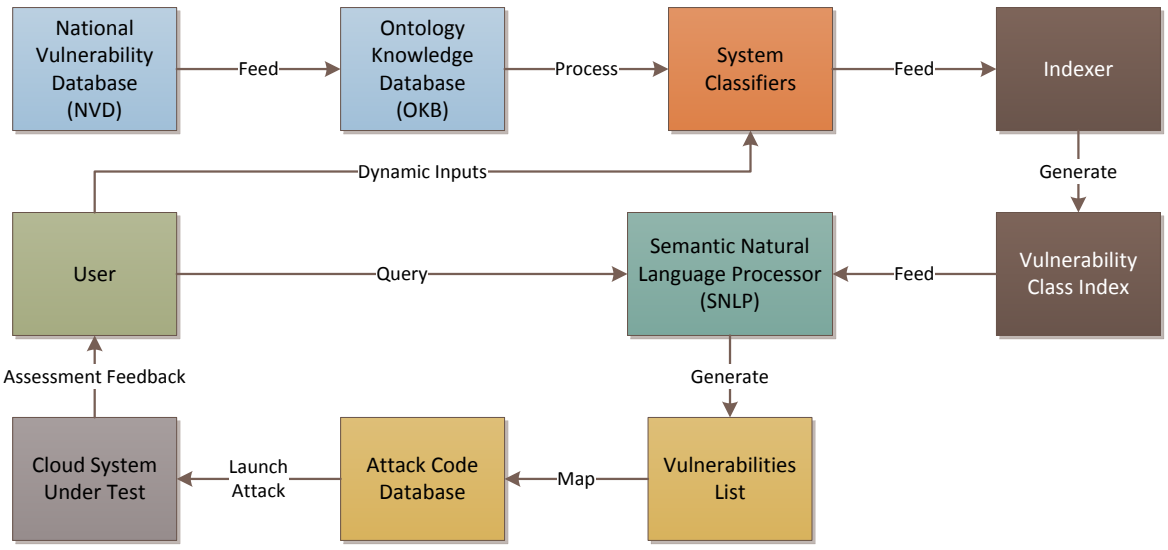

Fig. 2. High Level View of our VULCAN Working Process

CVE-2009-2692.

3) The SNLP component, will do reasoning tasks on the user query and using the created vulnerabilities group data. It will return to the user via a dialogue agent interface relevant results such as the IT Products that have vulnerabilities and other necessary information that comply with the user query.

4) Using our Middle-ware application, we map the found IT Products to a Mercury framework [2] module called "Test for vulnerabilities that allow a malicious application to gain root access" to launch attacks on the products within our targeted android user device.

5) Then, VULCAN traces the deployment of the module payloads and report whether the attacks were successful on the device or not and if the tested vulnerabilities are still present or fixed for those IT Products.

V. IMPLEMENTATION

We have implemented our VULCAN via a set of interconnected components as described above in the Architecture section. The main source vulnerability information for our framework is provided by NVD. The NVD data is stored in a hierarchical database which lacks reasoning on its data. In our implementation of OKB we extract NVD data and store them in a graph database which is realized via Resource Description Framework (RDF) triples. With our graph database we generate an ontology that enable us to do some reasoning tasks which are useful for vulnerability assessment within our VULCAN . To achieve a dynamic vulnerability assessment for Cloud Computing, we propose three modules such as: System Classifiers, Indexer, and Vulnerability Class Index. Each module depends on the other one as described in the Architecture section. In our SNLP implementation, we rely on our Ontology Knowledge Base for information and the capabilities of our modules to properly fetch the cloud computing relevant search results.

A sample demonstration of our VULCAN components and modules implementation is shown in [23].

A. OKB

We defined a vulnerability ontology to model vulnerability information provided by NVD. In our approach, we extended the ontology proposed in our previous work on Vulnerability Assessment In Cloud Computing [4]. This new ontology in Fig. 3 is more expressive in terms of new entities and relationships, we added a class (CloudType) and sub-classes to help us model cloud environment and its types and also to model in the Software subclass of ITProduct class which vulnerable programs are privileged or unprivileged. We implemented this ontology in Protege [24] and the source code is available in our demonstration set samples [23].

Ontology Knowledge Base (OKB) Implementation is completed via these two steps: *1*) Extraction of vulnerability information from the National Vulnerability Database (NVD) - XML data feed; and *2*) Population of our ontology knowledge base (OKB)

1) Extraction of vulnerability information from NVD - XML data source
   a) Parsing the NVD - XML data source
   b) Extracting from XML feed each entry relevant attributes for examples:
      i) CVE-ID
      ii) IT-Products
      iii) CVSS-Metrics
      iv) Summary
   c) From the extracted Summary text, another extraction take place to retrieve additional information (that was not provided in any entry's attribute of NVD) about this vulnerability described in the Summary text. such additional information are like:
      i) Who's the attacker
      ii) What's the attacker's intent
      iii) What's the attack mechanism

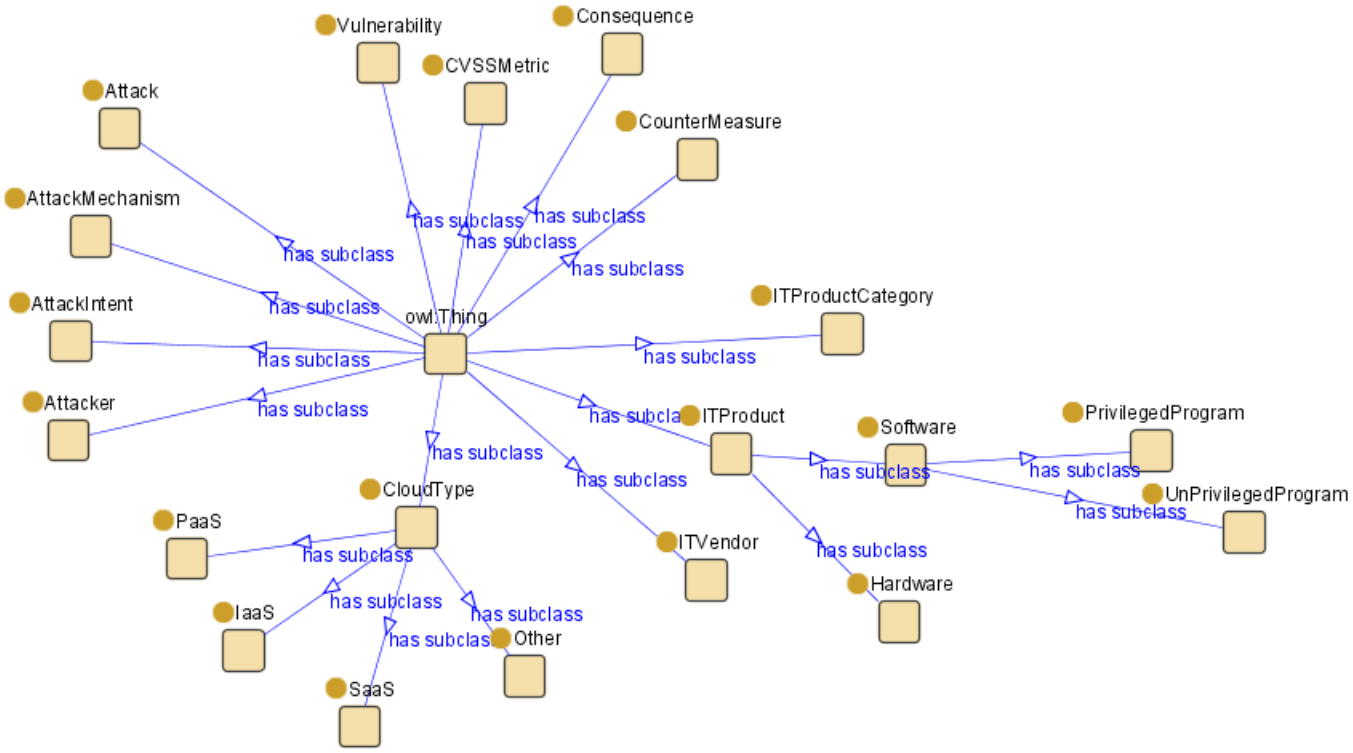

Fig. 3. High Level View of our Vulnerability Ontology Definition

d) Map the CVE-ID extracted in (b) to our web search agents to retrieve additional information about this particular vulnerability. Such information are like:
   i) What is the attack (exploit)?
   ii) What is the consequence of the attack extracted in (c)?
   iii) What is the countermeasure of this attack (c)?

2) Population of our OKB
   a) Using Protege-OWL editor [24], we first define our vulnerability ontology domain in terms of concepts (classes), roles(properties, relationships) and individuals [8].
   b) Then we populate our ontology to create a knowledge base of vulnerabilities. We use these two adopted approaches:
      i) Manually extract relevant vulnerability information from NVD - data source and use them to instantiate our ontology.
      ii) Using custom python script, we automatically extract relevant vulnerability information as described in Step-1.
   c) Then we store them into a triple store database. This database will be used to instantiate our ontology via Protege.

In the OKB process, we implemented our extractors using custom python scripts. These extractors, they iteratively retrieve relevant vulnerability information from each NVD entry. With the extracted data, we generate RDF triples using an RDFLIB [25] python library. With that we populated our defined ontology automatically.

For a small set of NVD data entries, one can use Protege tool to achieve the same goal. By manually creating the ontology instances. In Protege, the ontology population can be achieved either by adding instances one at a time or by instantiating them using a backend database.

B. Modules

Our modules for the VULCAN implementation as illustrated in Fig. 4 are: System classifiers, Indexer and Vulnerability class index.

1) System Classifiers: Our proposed approach for the system classifiers implementation is illustrated in our modules Fig. 4. We are customizing an application of genetic algorithms that are more adaptive to our dynamic inputs for cloud computing classification. Using the properties of genetic algorithms of working on a population of possible solutions and being stochastic, we rely on them on generating some classes that are then feed to our indexer module for further processing.

2) Indexer: The indexer module application will use the feeds received from system classifiers module to browse our vulnerability ontological knowledge base. As the module illustrates in Fig. 4, our indexer application should repeatedly check for any new change in the provided dynamic inputs and creates new indexes. Our goal for the implementation of this module is to optimize speed and performance in finding relevant information for the SNLP search queries.

We first collect the classes generated by the system classifiers, then use them to parse our OKB component into groups that are related to the provided feeds. Then we store the indexes as linked data. This approach will allow us to do inference on SNLP search results.

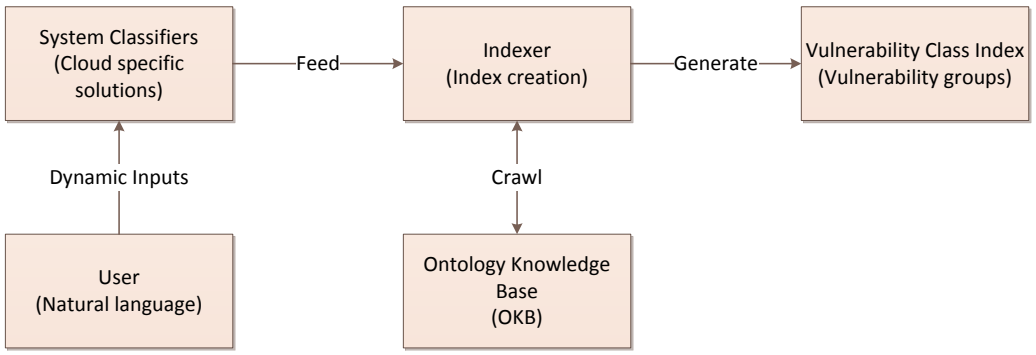

Fig. 4. VULCAN Modules

3) Vulnerability Class Index: The indexes created by our indexer module, are further processing and listed into vulnerability class groups as illustrated in Fig. 4. These groups reflect the cloud based dynamic inputs received. Then, within our VULCAN framework, the SNLP component uses these rich information about vulnerability for retrieving results for the relevant user given query. The implementation of this module is straight forward, all it needs to point an extractor to the indexed data and retrieve them as a list.

C. SNLP

Our Semantic Natural Language Processor engine enables users to search and reason about vulnerabilities via these interconnected modules:

1) pattern matching
   a) This technique helps us to identify any kind of pattern from the user input text. We realize it using Regular Expression methods. Here we both do a keyword search and reasoning, then formalize a suitable result to respond the user query.
2) keyword search
   a) Queries protege plugin in [24] allows the user to query our vulnerability ontology using a plain keyword, or by selecting a class (concept) or relationships name within our ontology. In addition to the search results, user to learn more about related information via generated inline links.
3) reasoning
   a) To reason with our ontology knowledge base stored in an owl file for example, we use two methods. One method is by using the SWRL Protege plugin [26], here the user enter the SWRL rules via an editor to reason about owl individuals and to infer new knowledge about them. Another approach is to use Jess [27] as the rule engine to achieve the same goal as the SWRL plugin does.
   b) Pellet [28] is one of the reasoner tool we could use for OWL-DL [29] reasoning tasks.
   c) SPARQL query [30] allow us to query our RDF format ontology and perform some reasoning tasks.

Our SNLP component as illustrated in Fig. 5 is a self-contained application that allow the user to lively interact with a given system (in this case, our VULCAN) via a its dialogue agent interface. Here, the user input a query which can be a formal one (like a SPARQL query) or not. Then it is processed through our engine processor which run a pattern matching, keyword search and reasoning tasks while generating partial results. A formalized user query result is produced from one or a combination of the partial results. This application is implemented using a similar approach as the intelligent personal assistant and knowledge navigator system uses like in SIRI [31]. Here we interlock our OKB and modules (System Classifiers, Indexer, and Vulnerability Class Index) together to support our intelligent system. In order to be able to produce a reliable and relevant result of the user's query.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced VULCAN , a Vulnerability Assessment Framework for Cloud Computing. In the effort to model security vulnerabilities, we defined a vulnerability ontology that classifies them. Then, we developed an automated process to instantiate our ontology using the data provided by NVD which resulted into our ontology knowledge base (OKB). Using this rich OKB, we are able to study and assess security vulnerability of individual or component parts of the cloud environment system. We achieve this complete assessment via VULCAN components, such as Semantic Natural Language Process (SNLP), and modules, like System Classifiers and Indexer.

We envision that cloud computing users, providers, security analysts can use VULCAN features to perform different type of assessment of their cloud environment. Also, our framework is flexible that developers can extend it by creating and adding new modules and components as they see fit. In addition, user's can integrate our VULCAN 's capabilities into any other compatible mobile, desktop or cloud security assessment frameworks.

Currently we have a prototype implementation of our VULCAN framework. We plan to extend its features as part of Future work. First goal is add metrics into our framework that allows users to compare different vendors, products, infrastructure based on the presence or absence of known vulnerabilities. The second goal is add more information in the OKB that allows user's to identify relationships among vulnerabilities. We also plan to explore the deployment of our

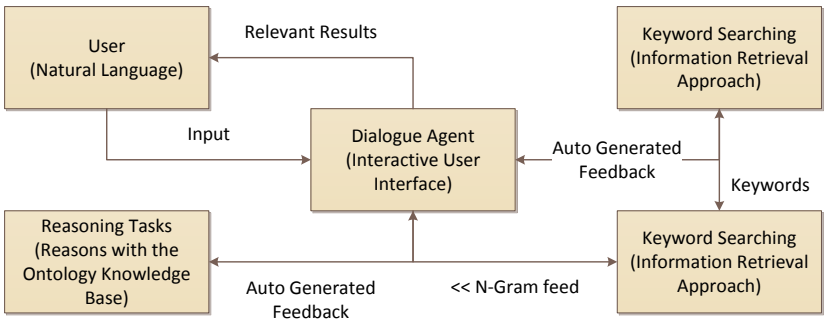

Fig. 5. Semantic Natural Language Processor

SNLP component for mobile devices as an application that enable user to assess vulnerabilities on-the-fly.

Ultimately, VULCAN shroud be able to mitigate current threats that face cloud environment by its known vulnerabilities. Our framework is capable of exposing those vulnerabilities individually and also for a given cloud system target, we should be able to discover new possible vulnerabilities by performing reasoning tasks.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] NICOLE PERLROTH and QUENTIN HARDY. *Bank Hacking Was the Work of Iranians, Officials Say*. The New York Times. 2013. URL: http://goo.gl/IIXvt.

[2] T. Erasmus. "The heavy metal that poisoned the droid". In: (2012).

[3] A. Steele. "Ontological Vulnerability Assessment". In: *Web Information Systems Engineering–WISE 2008 Workshops*. Springer. 2008, pp. 24–35.

[4] S. Kotikela, K. Kavi, and M. Gomathisankaran. "Vulnerability Assessment In Cloud Computing". In: ().

[5] *National Vulnerability Database*. NIST. 2012. URL: http://nvd.nist.gov/.

[6] *Metasploit Auxiliary Module and Exploit Database (DB)*. Metasploit. 2012. URL: http://www.metasploit.com/modules/.

[7] M. Guo and J.A. Wang. "An Ontology-based Approach to Model Common Vulnerabilities and Exposures in Information Security". In: *ASEE Southest Section Conference*. 2009.

[8] J.A. Wang and M. Guo. "OVM: an ontology for vulnerability management". In: *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. ACM. 2009, p. 34.

[9] R. Paul, I.L. Yen, F. Bastani, J. Dong, W.T. Tsai, K. Kavi, A. Ghafoor, and J. Srivastava. "An Ontology-Based Integrated Assessment Framework for High-Assurance Systems". In: *Semantic Computing, 2008 IEEE International Conference on*. IEEE. 2008, pp. 386–393.

[10] J.A. Wang, M. Guo, H. Wang, M. Xia, and L. Zhou. "Ontology-based security assessment for software products". In: *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. ACM. 2009, p. 15.

[11] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie. "Automated Extraction of Security Policies from Natural-Language Software Documents". In: (2012).

[12] N. Yahia, S.A. Mokhtar, and A.W. Ahmed. "Automatic Generation of OWL Ontology from XML Data Source". In: *arXiv preprint arXiv:1206.0570* (2012).

[13] I. Bedini and B. Nguyen. "Automatic ontology generation: State of the art". In: *PRiSM Laboratory Technical Report. University of Versailles* (2007).

[14] P. Meunier. "Classes of vulnerabilities and attacks". In: *Wiley Handbook of Science and Technology for Homeland Security* (2008).

[15] T. Vidas, D. Votipka, and N. Christin. "All your droid are belong to us: A survey of current android attacks". In: *Proceedings of the 5th USENIX conference on Offensive technologies*. USENIX Association. 2011, pp. 10–10.

[16] O. Sheyner and J. Wing. "Tools for generating and analyzing attack graphs". In: *Formal methods for components and objects*. Springer. 2004, pp. 344–371.

[17] T. Heberlein, M. Bishop, E. Ceesay, M. Danforth, CG Senthilkumar, and T. Stallard. "A Taxonomy for Comparing Attack-Graph Approaches". In: ().

[18] *Security Content Automation Protocol*. NIST. 2012. URL: http://scap.nist.gov/.

[19] *Common Vulnerabilities and Exposures*. MITRE. 2012. URL: http://cve.mitre.org/.

[20] *Common Weakness Enumeration*. MITRE. 2012. URL: http://cwe.mitre.org/.

[21] *Common Platform Enumeration*. MITRE. 2012. URL: http://cpe.mitre.org/.

[22] *Common Vulnerability Scoring System*. FIRST. 2012. URL: http://www.first.org/cvss.

[23] *VULCAN*. Trusted Secure Systems Lab, University of North Texas. 2013. URL: https://github.com/vulcan13/VULCAN.

[24] *welcome to protege*. National Library of Medicine. 2012. URL: http://protege.stanford.edu/.

[25] Daniel Krech. *RDFLib*. 2012. URL: http://en.wikipedia.org/wiki/RDFLib.

[26] Martin O'Connor. *SWRL Tab*. 2012. URL: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab.

[27] *Jess,the Rule Engine for the Java Platform*. Ernest Friedman-Hill. 2012. URL: http://herzberg.ca.sandia.gov/.

[28] Clark and Parsia. *Pellet Reasoner Plug-in for Protege 4*. 2012. URL: http://clarkparsia.com/pellet/protege/.

[29] *OWL Web Ontology Language Guide*. W3C. 2012. URL: http://www.w3.org/TR/owl-guide/.

[30] *SPARQL Query Language for RDF*. W3C. 2012. URL: http://www.w3.org/TR/rdf-sparql-query/.

[31] *Siri*. Apple Inc. 2012. URL: http://www.apple.com/ios/siri/.