# Automated Event-driven Security Assessment

by

Jeong-Jin Seo

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved January 2014 by the
Graduate Supervisory Committee:

Gail-Joon Ahn, Chair
Stephen S. Yau
Joohyung Lee

ARIZONA STATE UNIVERSITY

May 2014

# ABSTRACT

With the growth of IT products and sophisticated software in various operating systems, I observe that security risks in systems are skyrocketing constantly. Consequently, *Security Assessment* is now considered as one of primary security mechanisms to measure assurance of systems since systems that are not compliant with security requirements may lead adversaries to access critical information by circumventing security practices. In order to ensure security, considerable efforts have been spent to develop security regulations by facilitating security best-practices. Applying shared security standards to the system is critical to understand vulnerabilities and prevent well-known threats from exploiting vulnerabilities. However, many end users tend to change configurations of their systems without paying attention to the security. Hence, it is not straightforward to protect systems from being changed by unconscious users in a timely manner. Detecting the installation of harmful applications is not sufficient since attackers may exploit risky software as well as commonly used software. In addition, checking the assurance of security configurations periodically is disadvantageous in terms of time and cost due to zero-day attacks and the timing attacks that can leverage the window between each security checks. Therefore, event-driven monitoring approach is critical to continuously assess security of a target system without ignoring a particular window between security checks and lessen the burden of exhausted task to inspect the entire configurations in the system. Furthermore, the system should be able to generate a vulnerability report for any change initiated by a user if such changes refer to the requirements in the standards and turn out to be vulnerable. Assessing various systems in distributed environments also requires to consistently applying standards to each environment. Such a uniformed consistent assessment is important because the way of assessment approach for detecting security vulnerabilities may vary across applications and operating systems.

In this thesis, I introduce an automated event-driven security assessment framework to overcome and accommodate the aforementioned issues. I also discuss the implementation details that are based on the commercial-off-the-self technologies and testbed being established to evaluate approach. Besides, I describe evaluation results that demonstrate the effectiveness and practicality of the approaches.

TABLE OF CONTENTS

Page

# 1. INTRODUCTION

## 1.1 OVERVIEW

The government, defense, and private sectors have been struggling to keep computer systems away from security breaches. Among many useful methods to secure systems, *Security Assessment* has been considered as an effective method to measure assurance of systems based on security standards and the status of compliance with baselines [32]. Such standards and baselines could help systems avoid well-known risks and describe the weakest points of systems by allowing rigorous security analysis and discovering configurations that cause potential risks in systems. Furthermore, Security assessment enlightens parties to understand security goal precisely, and thus it may mitigate security risks and ensure an appropriate level of system assurance.

US federal government has recognized the importance of security assessment and started to develop plans for using Microsoft's operating system based on specific security configurations, which stem from US Air Force's common security configuration for Windows XP. This initiative was expanded to deal with other operating systems such as iOS, Linux, and HPX [33][34]. The Office of Management and Budget (OMB) and National Institute of Standards and Technology (NIST) developed Federal Desktop Core Configuration (FDCC) baseline for the purpose of security management, which prevents security problems as early as possible caused by malfunctioned operating system and faulty configurations which have been found by security testers or attackers [33]. The United State Government Configuration Baseline (USGCB) evolved from FDCC replaces the FDCC baseline for Information Technology (IT) products widely deployed across agencies [34]. However, under increasing number of products and various operating systems, establishing standards and measuring security risks are getting harder to collect and test all of security configuration resources by security analysts. The FDCC and USGCB teams decided to work together with organizations and companies for collecting vulnerability information with XML-based well-structured format and maintain information into one repository to facilitate easy sharing process and comprehensive protection. As a result, the XML format of vulnerability information has been discussed, analyzed, stored, and disseminated by central place, MITRE Corporation. This XML-based specification was named as Open Vulnerability Assessment Language (OVAL) for the purpose of a single security standard that is both human- and machine-readable and covers various operating systems and its applications [20][21].

The main purpose of both FDCC and USGCB is to develop and implement security configuration baselines, and gather security assessment results to provide the current status of system assurance to stakeholders. In other words, it allows them to determine how much security problems could be occurred in a target system with the help of security baselines. However, both departments mainly focus on the detection of vulnerabilities in the system. Consequently, Federal Information Security Management Act (FISMA) was initiated to prioritize risk-based security assessment and real-time continuous monitoring of security controls as a critical focus of compliance and security, due to a dramatic increase in security incidents at federal agencies [19]. For example, the reported security incidents among 24 key agencies increased more than 650% in the last five years but ironically these federal agencies have periodically performed security assessment based on security configuration baselines.

Therefore, the need of new continuous security monitoring that depends on system environments has been recently addressed in the security community. The previous monitoring approaches have been mainly dedicated to share security incidents by stacking security issues up in the repository as much as possible. By using the gathered information, the traditional security monitoring approach periodically measures system assurance since it is tedious and costly to evaluate the entire system with various standards. However, it is critical to perpetually monitor the previously discovered security issues in a target system since it would be worthwhile to detecting new risks. Every events in the target system with respect to users' behavior such as installing software packages or patching updates should be considered to perform security assessment so that we can achieve a more comprehensive assessment to reflect any changes and modifications on the system's configuration. Thus, the periodic security assessment is not sufficient enough to measure the assurance of systems.

We reiterate that we need event-driven continuous monitoring system is necessary to consider the newly changed configuration that may draw security problems on the system. Without evaluating system environments reflecting to the system configuration that can be frequently changed by users, continuous security assessment is the most effective approach to reduce and eliminate potential risks. In addition, as mentioned above, diverse environments and various software applications that change system configurations are another obstacle to hinder the accurate assessment of systems. To perceive comprehensive security risks is increasingly tough even to security experts because it is difficult to understand or define different domains of security problems. We thus need a comprehensive and domain-independent approach that can be used in multiple environments in a seamless manner. There are many commercial tools to discover and share vulnerability information to the public for the purpose of avoiding security risks. Without having the integrated data to provide the commonly understandable security information to each computer environment, it is also hard to measure system assurance comprehensively. In this thesis, we leverage the notion of ontology to build a system that can support various environments for performing efficient high-level reasoning and making better decision.

## 1.2 STATEMENT OF THE PROBLEM

There are several questions that this thesis attempts to address. First, Security Assessment (SA) is crucial part of measuring assurance status of systems, but most SA techniques have been focused mostly on how to define and detect vulnerability or vulnerable configurations with the periodic system check. Security administrator performs security assessment periodically because it is obligated to comply with the security standards and it helps discover inappropriate configurations in the system and the potential issues that can be missed without deep inspection of the system. So, it might be enough to realize current system status but the drawback of periodic security measurement is that configurations could be changed any time by users. Event-driven security assessment is strongly demanded because of this reason. For example, suppose a security administrator sets the security measurement task up for twenty-four hours and a user changes one of system configurations an hour after the measurement was performed. If the changed configuration meets the conditions of vulnerable configurations, then attackers can still have twenty-three hours to use this security configuration breach for their malicious purpose. By this reason, event-driven continuous monitoring should be carried out.

Second, applying proper security standards corresponding to each system helps systems keep safe. Many companies and organizations generate and provide OVAL-based security assessment practices to the public. Many standards with various approaches help understand a wide-range of security issues. However, it is not even easy for security administrators to determine which standards should be applied in a target system considering the characteristics of a system environment since computer configurations could be different based on users' preferences of operating systems and applications. And each standard has different perspectives to interpret configurations so it is not manageable without having comprehensive understanding on each standard. So, it is fairly a time consuming task to know which standard should be applied properly in a target computer and how to apply it. Also, it is necessary to have data not only consolidating security information for the system, but also capturing characteristics of different environments properly. Moreover, the structure of data should be expandable since security risks in the system keep growing continuously.

Third, the environments of a system can vary based on the role and services that the system provides. There exist many operating system dependent security assessment tools. However, more intuitive but system-independent security assessment is required. By using system-level implementation, tool is applicable to the various environments in a seamless manner. This tool provides security assessment consistency for diverse environments.

Last but not least, security assessment with the specific viewpoint of security administrators is more effective since their interests on a particular aspect of vulnerabilities in a system would help clearly recognize current risks and its affects to the system. In other words, providing user-centric security assessment helps security administrators monitor security gaps between security countermeasures and their point of view on vulnerabilities.

## 1.3 OUTLINE OF THE THESIS

The thesis is organized as follows. Chapter 1 addresses motivation of this work and problem statements including the overview of security assessment standard and the importance of event-driven comprehensive security measurement system, followed by the related work in Chapter 2. In Chapter 3, we also overview background technologies that are leveraged to realize the proposed security assessment approach. Chapter 4 describes an event-driven continuous monitoring framework and elaborates each component in our framework. Furthermore, we show the architecture of system-independent event-driven monitoring system. The implementation details including algorithms and evaluation of our system are discussed in Chapter 5 and Chapter 6 concludes this thesis along with the contributions and future works.

# 2. REALTED WORK

Risk assessment has been part of core security methods. Most risk assessments have been performed with risk analysis and monitoring. While evaluating security disciplines, applying undifferentiated security disciplines is not straightforward since each environment has its own nature. Also, the security administrators who analyze vulnerabilities existed in the system may want to see analysis results based on their preferences. In this thesis, we focus on an event-driven system analysis approach to identify risks and then show results in accordance with the preferences of the security administrators. To achieve this, we first review relevant methods that we leverage in this thesis including Security Information and Event Management (SIEM), Common Information Model (CIM), and ontology. To accomplish event-driven risk assessment in different environments, we introduce continuous monitoring system that can work under various system settings. We then discuss the integrated security requirement framework and risk assessment method to check security compliance.

Many companies have adopted Security Information and Event Management (SIEM) and introduced real time tools to mainly identify systems' weaknesses by investigating system configurations based on security policies and compliance requirements. Previous government reports show that proper review of vulnerability and SIEM had been done early, but the correlation between continuous monitoring and SIEM has not been achieved. Security assessment has been rather periodically performed so far [2].

Most organizations have to patch and configure their products for the security reason and their products are maintained by the security postures at any given time to keep the systems safe. Furthermore, organizations are obligated to be compliant with sets of security requirements. To support such a critical obligation, Security Content Automation Protocol (SCAP) was introduced and published by NIST [3]. To avoid any unnecessary steps in security assessment, SCAP works with OVAL. By taking advantages of SCAP, maintaining enterprise systems, inspecting system security configuration settings, and examining signs of potential compromises in the systems have been extremely efficient [4]. SCAP can collect vulnerability information from different vendors and integrate information into definitions that contain checking methods so that security administrators can examine security risks with a given set of compliance requirements. The current version of SCAP performs measurement of system assurance and monitoring of security setting [5]. The SCAP uses top-down approach for the measurement and OVAL is the main step of the assessment process, which contains security contents about the way to measure a specific machine's state associated with system details. Based on this system details, OVAL generates assessment results by expressing the state of each machine. To achieve goal of sharing information, OVAL enforces structural standard but it cannot provide flexible measurement because of this structural dependency.

CIM and Web-Based Enterprise Management (WBEM) architecture [29] are another related work. There are many approaches that took advantage of CIM and WBEM to achieve their security goal [25]. In these approaches, the CIM is mostly used to collecting and gathering data from operating system configuration. Also, CIM is utilized to retrieve data from a system and provide such data to check the current security status in the system. Even though these approaches resulted in an effective set of security controls and risk management process, it may increase the burdens of data management since data storage can be quickly filled due to the infinite number of events incautiously caused by end users. Such events might cause security breaches in the system so each reflected data should be compared or matched to the overall security standard.

The ontology represents a set of relational concepts within domain and the relationships among its concepts of domain can be represented with CIM. In other words, CIM defines classes and relations can be represented by ontology [29]. There exist several research approaches to make connection between system information and security features, using both ontology and CIM at the same time [35].  However, making ontology and expanding the data relevant to vulnerable information still need to be studied. . Especially, gathering information by CIM and generating ontology based on the gathered information are costly. Suppose we deal with the cloud-based environments and there exist many different configurations in the virtual machine (VM) depending on users' preferences. Under such environments, ontology has to cover all configuration changes in each VM but it will be a time-consuming task. Therefore, it is necessary to develop systematic procedures for leveraging CIM and ontology to represent vulnerabilities in a more effective manner,

Government agencies and organizations started to focus on developing continuous monitoring systems. As a result, the Federal Network Security (FNS) Branch of Department of Homeland Security launched the Continuous Asset Evaluation, Situational Awareness, and Risk Scoring (CAESARS) [8]. The objective of their project is to build a concrete vendor-neutral architecture and incorporate the main elements of the Department of State (DOS).

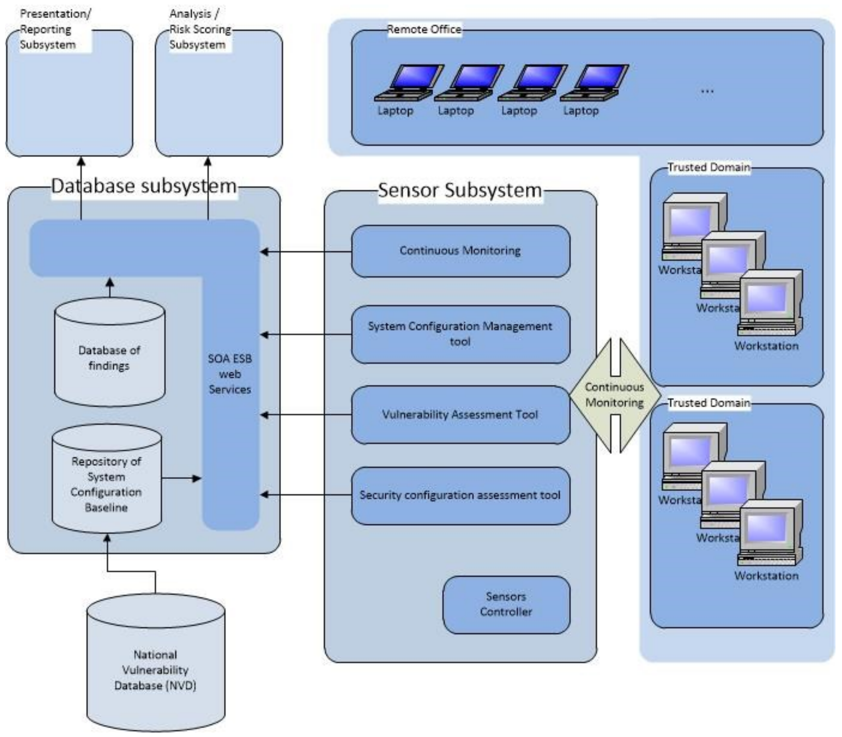

Presentation/ Reporting Subsystem
Analysis / Risk Scoring Subsystem
Remote Office
Laptop
Laptop
Laptop
Laptop
...
Database subsystem
Sensor Subsystem
Trusted Domain
Continuous Monitoring
Database of findings
SOA ESB web Services
System Configuration Management tool
Vulnerability Assessment Tool
Continuous Monitoring
Trusted Domain
Repository of System Configuration Baseline
Security configuration assessment tool
Workstation
Sensors Controller
National Vulnerability Database (NVD)

Figure 1: Conceptual Description of the CAESARS System

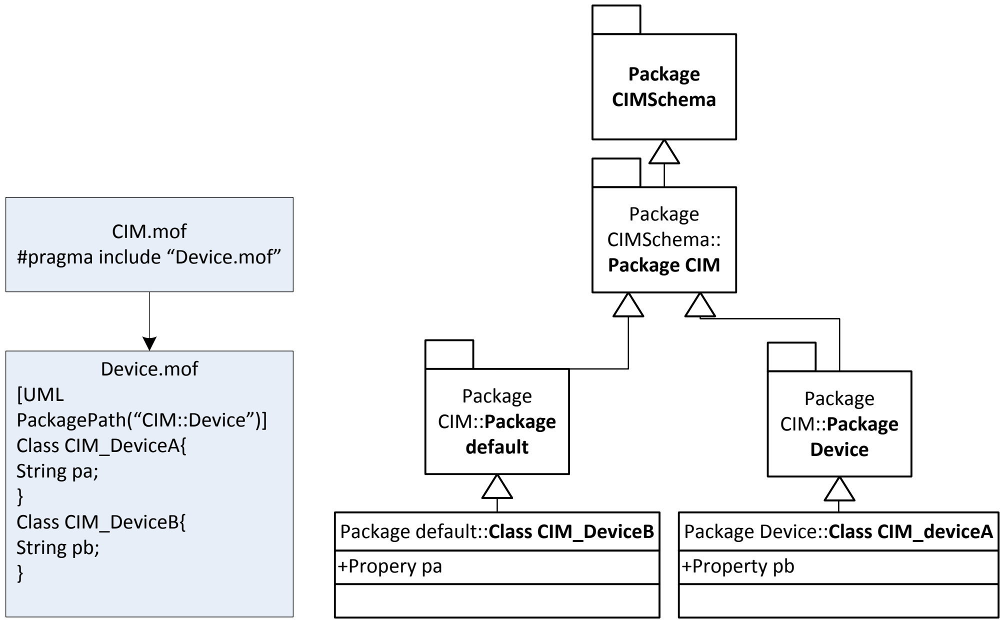
CAESARS system has integrated security postures with determining the gaps between current state and security baseline and ensuring that the every system and application does not contain tested potential security problematic configurations. For ensuring that every system meets security policies and compliance requirements, CAESARS system provides four subsystems as shown in Figure 1: sensor subsystem, database/repository subsystem, analysis/risk scoring subsystem, and presentation and reporting subsystem.

# 3. BACKGROUND TECHNOLOGY

## 3.1 COMMON INFORMATION MODEL

Distributed Management Task Force (DMTF) published the CIM standard to exchange management information about managed elements that is the structure of the information contained among multiple parties. By using CIM, software, which manages information, does not need to be written again for converting operations or information since CIM attempts to unify and extend the existing instrumentation and management standards using Object-Oriented Constructs and Design (OOD) [26]. CIM model leverages OOD-based techniques to have richer representation of management data. The architecture of CIM is convertible to Unified Modeling Language (UML) which can be represented between CIM classes and CIM associations, either ways. So, the CIM can not only describe classes and its relationship among classes of objects, but also enables to have various relationships with other managed elements. The CIM is composed of two parts: CIM *infrastructure specification* and CIM *schema*. The CIM infrastructure specification provides managed elements and its relationships by allowing specialization of common base elements to access specific features of the system. The system needs to provide its information as an object through the CIM managed elements. The CIM schema is a conceptual schema which enables the CIM client to communicate with managed elements in a system. CIM schema covers most elements in the computer product, such as computer systems, operating systems, networks, middleware, services and storages. The strength of CIM schema is that it can be extended seamlessly with the common functionality defined in CIM schema.

Users can specify, visualize and document software systems using UML from the Object Management Group (OMG) [9]. The UML-based specification is converted to the corresponding CIM MOF file and vice versa. The following example is a package for the mapping between CIM MOF file and UML elements.



CIM.mof
#pragma include "Device.mof"
Device.mof
[UML
PackagePath("CIM::Device")]
Class CIM_DeviceA{
String pa;
}
Class CIM_DeviceB{
String pb;
}
Package CIMSchema
Package CIMSchema:: Package CIM
Package CIM::Package default
Package CIM::Package Device
Package default::Class CIM_DeviceB
+Propery pa
Package Device::Class CIM_deviceA
+Property pb

Figure 2: Mapping between CIM MOF file and UML elements

In Figure 2, the *CIM_DeviceA* has the *UMLPackagePath* qualifier, so its value gets information under a target package path of a device as UML elements shows. For the *CIM_DeviceB* class, the *UMLPackagePath* is not specified so the default *UMLPackagePath* is applied and vice versa. This is a simple example that shows how CIM schema is applied to MOF files. UML package whose package path under a target package shall own the UML class which a CIM class is mapped with the inheritance. This general mapping between CIM MOF and UML elements allows CIM to support any computer environments.

### 3.1.1 WINDOWS MANAGEMENT INSTRUMENTATION

Windows Management Instrumentation (WMI) is a set of extensions to the Windows Drive Model (WDM), which is the framework for device drivers that provide system interfaces to provide information and notification based on CIM and WBEM. WMI enables to managing windows-based personal computers both locally and remotely by Desktop Management Interface (DMI), which is a standard framework that tracks and manages components in desktop, laptop or server. By leveraging existing management applications, WMI can also generate and provide comprehensive management as a uniform and reference model by acquiring management data from various heterogeneous sources in a common way.

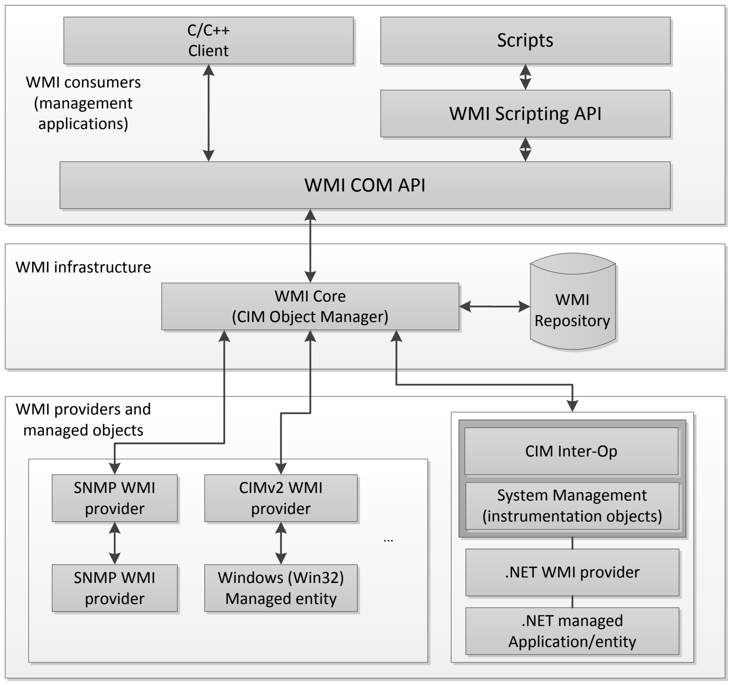

C/C++ Client
Scripts
WMI consumers (management applications)
WMI Scripting API
WMI COM API
WMI infrastructure
WMI Core (CIM Object Manager)
WMI Repository
WMI providers and managed objects
CIM Inter-Op
System Management (instrumentation objects)
SNMP WMI provider
CIMv2 WMI provider
...
SNMP WMI provider
Windows (Win32) Managed entity
.NET WMI provider
.NET managed Application/entity

Figure 3: WMI Architecture

The main components of WMI architecture stem from CIM components. Those components are WMI provider, the CIM object manager (CIMOM) and CIM repository as illustrated in Figure 3.

WMI providers monitor and communicate with physical and logical system components made up with operating system services and utilities, hardware and applications. WMI providers are an extension of WDM and send its data information into WMI repository with the managed format described in MOF files. These providers mainly provide information as a set of managed objects in response to the requests coming from CIMOM received in a WMI consumer. The MOF files can be compiled by MOF compiler in WMI and added into WMI repository for the managed data.

CIMOM manages the data transfer among WMI providers, the CIM repository, and management applications. The procedure of transferring data is made in the following steps: the WMI provider retrieves information from resources and CIM repository stores information requested by WMI consumer layer. CIMOM creates indication subscription in the CIM repository and contacts WMI provider to receive the requested information from clients. The CIMOM sends the received information from the provider to the WMI consumer. The data can be manipulated by WMI Query Language (WQL), which is written in a SQL-like format. And WMI has a function to notify events coming from the provider both locally and remotely. WMI event notification is capable of monitoring the state of the systems across the network. There are two kinds of event notification: synchronous and asynchronous event notifications. Synchronous event notification is paused until the method call returns the collection of objects. In contrast, asynchronous event notification allows continuous execution of WMI methods or provider methods while returns the collection of objects.

CIM repository is the storage to store the registered information that providers and applications provided with the managed format added in the repository by MOF files. The data in CIM repository can be easily out-of date, therefore, CIMOM executes queries to extract the changed data dynamically from the repository. This helps consumers receive the recent event information that providers give.

### 3.1.2 IMPLEMENTATION OF CIM IN WMI

WMI is an infrastructure to support CIM model and Microsoft Windows-specific extension of CIM. However, all schemas in the WMI repository are CIM-based schemas. Only "cimv2" namespace, which obtains data from Microsoft Win32 APIs, is CIM schema-based such as CIM core, system devices and application models. For example, Windows 7 introduced Win32_PowerPlan WMI class. This class resides in the cimv2 WMI namespace so that any script or code can trigger this information to receive power status of current machine from the client by executing WQL. When the CIMOM receives the request for information, the CIMOM checks an appropriate provider if the provider support dynamic data or notification of events of the requested information. If not, CIMOM forwards the request to the appropriate provider to return the requested information from resources. The return data format of WMI provider is described in CIM. The result format is standardized so any environment can use this data format to receive data and use it. There are many useful CIM classes--especially **CIM_RecordLog** class that can log and filter out other logs by names. By using this class, a system can derive notifications of event information from a provider. WMI already have running Win32 classes to record log files for the event so that system can get event information by using WQL. Using .NET framework, applications can be developed using data from WMI classes. It means the system can assess management information in an enterprise environment. If the provider does not exist in certain management information related to CIM, the system can create a provider based on CIM class and receive information from the created provider, which allows the system to access all WMI data.

## 3.2 ONTOLOGY

Knowledge sharing and reuse have many challenging issues [10]. The sharing and reuse of data is currently achieved but still lacks understanding of data semantics between entities. Sharing information in knowledge means the transfer from the sender to the receiver that could not use the same format for data representation in most cases. In this reason, extra care must be taken when the messages are transferred. The information is transferred in the way of structured format that is understandable to both sender and receiver. The message should be also transferred between sender and receiver who may use different formats. It means each party needs to process the transferred information on the knowledge base through the use of logical language. Moreover, the architecture of relational database does not represent $n : m$ relationship [27]. The additional table is needed to transform $n : m$ relationship into a $1 : n$ and a $1 : m$ relationships. This necessary step needs to be solved without schema modification. The lack of standard causes many unnecessary steps to share and reuse data between two sides. The effort to generate standardized results in the new way of sharing knowledge, ontology. Ontology can solve this problem by using formal and real-world semantics. Ontology provides formal semantics, which are machine and human understandable data format. Ontology attempts to detect every possible domain and support broad axioms for the expression of knowledge and it is ideally formal vocabularies shared by a group that is interested in a specific domain.

In the area of semantic web, ontology is used in various research fields such as knowledge engineering, database design, and information retrieval and extraction. The meaning of human understandable is that a word is in natural language and its relationships are reasonable to the human. The example of human understandable relationship is *is-a* relation, which denotes an association between super and sub concepts. The relationship describes the fact that one super concept is more general than another sub concept. The more general concepts are senior to the more specialized concepts in an is-a hierarchy as shown in Figure 4.

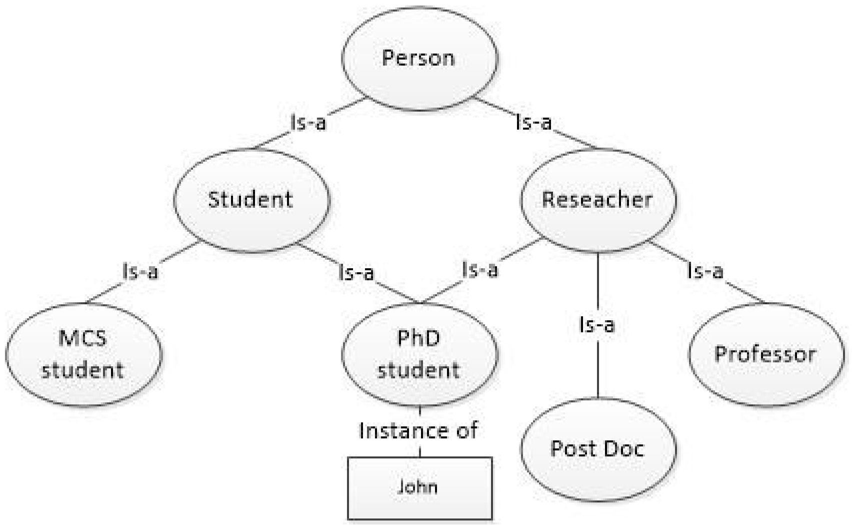


Figure 4: is-a hierarchy example

The relationship between entities may make many different conclusions. For example, both student and researcher can be a person. PhD student can be student and researcher, but MCS student cannot. This conclusion can be drawn by both computer and human since the formal nature of the relation can be explained respectively in this diagram. Real world objects can be described in the concepts. For example, John is instance of PhD student. The instance of relation means an actual concept derived by the PhD student. And all super concepts have *is-a* relation so that John must be an instance of the concepts such as PhD student, Student, Researcher, and Person. Ontology brings advantage of data and relation representation with several features such as flexibility and interoperability.

### 3.2.1 SPARQL PROTOCOL AND RDF QUERY LANGUAGE

The Resource Description Framework (RDF) [11] is the first language developed for the semantic web. RDF includes machine readable metadata to existing data on the web. RDF Schema (RDFS) [12] extends RDF with some basic (frame-based) ontological modeling primitives such as classes, properties and instances. Instance-of and subclass-of relationships are also introduced through RDFS. RDF has the object-attribute-value triple. It is commonly written as *(O, A, V)* [13][14]. Figure 5 is an example of RDF graph with this structure.

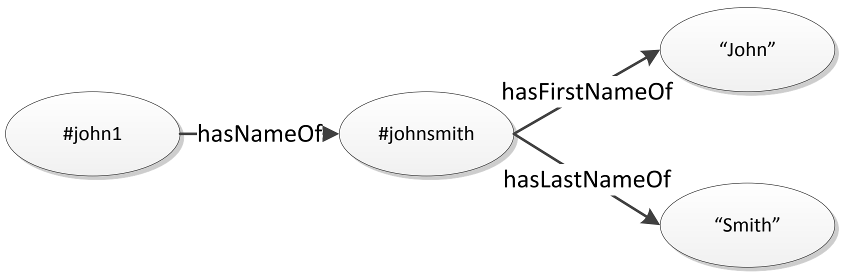


Figure 5: RDF graph example

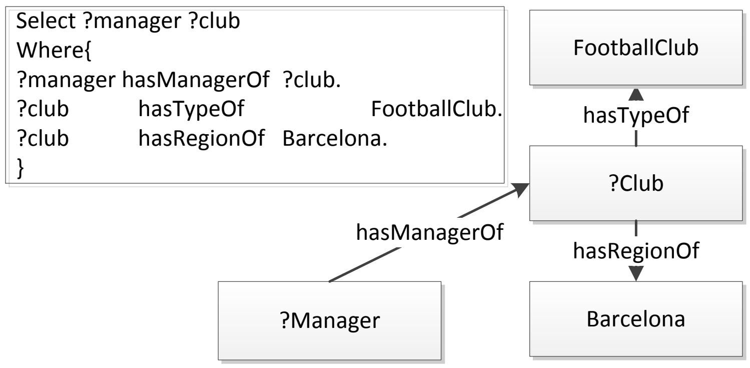
In Figure 5, an oval describes the resource and arrows that connect two resources show the predicate of the resource. The basic building block can be represented as follows:

```
(hasNameOf, #john1, #johnsmith)
(hasFirstNameOf, #johnsmith, “John”)
(hasLastNameOf, #johnsmith, “Smith”)
```

Figure 6: RDF triples example

RDF graph in Figure 5 is converted to RDF triples in Figure 6 with a predicate of each connection of resources. These simple three statements become very complicated in XML serialization. This is one of benefits that we can get from ontology. The XML schema describes how XML document ought to be ordered and combined in the predefined structure. In contrast, RDF schema does not describe the syntax of the RDF description, but the interpretation of each statement. This means RDFS defines classes and sub-classes for the class hierarchy, properties and its hierarchy. RDFS has the benefit of increasing formality of their subject and standard entailment of relationship among data.

The official W3C document describes SPARQL as follows [13]: “*Most forms of SPARQL queries contain a set of triple patterns called a basic graph pattern. Triple patterns are like RDF triples except that each of the subject, predicate and object may be variable. A basic graph pattern matches a sub-graph of the RDF data when RDF terms from that sub-graph may be substituted for the variables.”* SPAQL is generally graph matching execution [14]. For example, the query in Figure 7 returns the all football club that is based in Barcelona.



```
Select ?manager ?club
Where{
?manager hasManagerOf  ?club.
?club         hasTypeOf                  FootballClub.
?club         hasRegionOf   Barcelona.
}
```

Figure 7: SPARQL Query Example

In Figure 7, the query is written in the SPARQL query language and this example shows that it gets data set, which is strictly associated with two edges. One edge is 'hasTypeOf' which connects between club and *FootballClub* objects, and other is labeled as 'hasRegionOf', which is limited to the data set in Barcelona entity. The entities which match these conditions are allocated to the variable name of '?Club' and the manager, is returned if an entity meets the both edges. A simple SPARQL query can be converted into an SQL statement.

## 3.3 OPEN VULNERABILTY AND ASSESSMENT LANGUAGE

FDCC and USGCB published the checklist for checking vulnerabilities in the configuration of computer environments [15]. Security checklist is stored in National Vulnerability Database (NVD), which includes many kinds of security configuration including operating systems, applications and so on. The XML-based format for the checklists is specified in the Open Vulnerability and Assessment Language (OVAL) that is fundamental part to check the presence of vulnerabilities and configuration issues in a target system. This means that OVAL-based checklist called OVAL definition describes the technical details about security vulnerabilities and configurations in XML-based format. Security baseline in SCAP uses OVAL for checking baseline settings. OVAL is used to determine which vulnerabilities exist on a system and generate reports, and then system administrator deploys software patches or gets security countermeasures from assessment tools and takes proper actions based on organizational discipline or policies.

### 3.3.1 USAGE OF OVAL DEFINITION

OVAL is a standard to standardize the assessment information across the various security tools and services. The information security community has developed OVAL definitions by collaborating to create OVAL language and maintaining definitions in the OVAL repository from many participants and stakeholders. Industry, academy and government organizations try to share their vulnerability information through OVAL definitions. This effort helps share security issues and protects systems in a professional manner. OVAL works in three main steps: collecting characteristics from systems for testing, testing the presence of a machine state, and evaluating systems. For the collection of characteristics from a target system, it collects information of target system, system configurations, and other security relevant configurations in a standard XML format. By gathered system characteristics, assessment tool could receive vulnerability information associated with system. Any mismatched configurations will be eliminated or further examined. The standardized OVAL that encodes the vulnerability details of a specific machine state can check the system whether the system has any vulnerabilities, configuration setting meets the security policy, and patch is performed in the wide range of computer systems. There are many operating system based schemas to test a specific OS platform and its applications. Core schema and individual component schema tests basic and detailed system states of operating system platforms or applications, respectively. The result schema defines a standard XML format for generating an evaluation report. The report contains current configuration information of a system against OVAL definitions. The result schema allows administrators to compare the system with standards for verifying the existence of vulnerabilities or configurations which do not match security policies on the system

### 3.3.2 OVAL STRUCTURE AND ITS USE

The OVAL definition schema consists of two part of schema: core schema and a number of component schemas.

The core schema provides a structure of an OVAL definition to express metadata that is independent of an OVAL definition, which includes CVE identifier, platform under affected attribute, and description of the definition. Component schema is different from core schema and it defines the vulnerability, configuration and security issues within an OS platform and its applications.

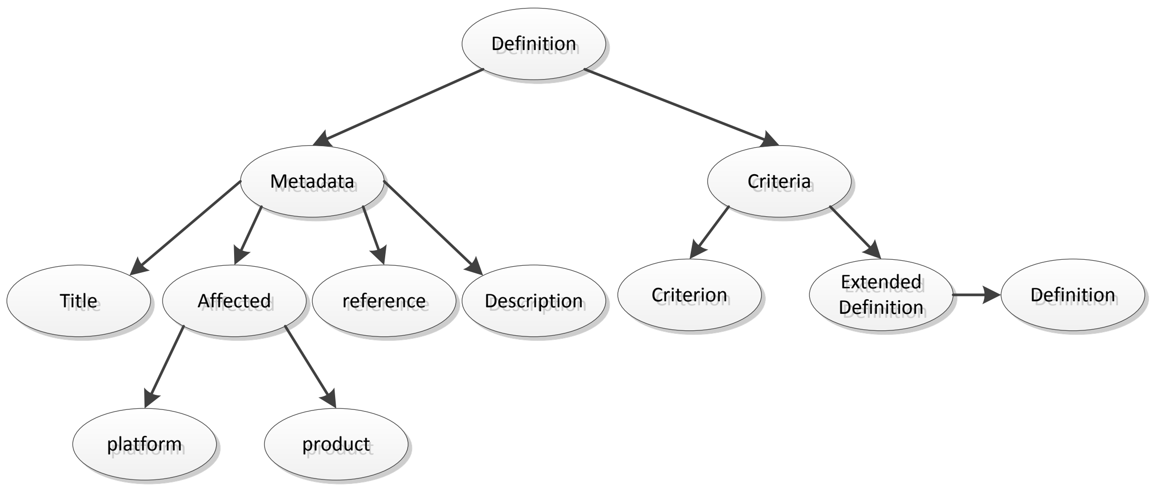
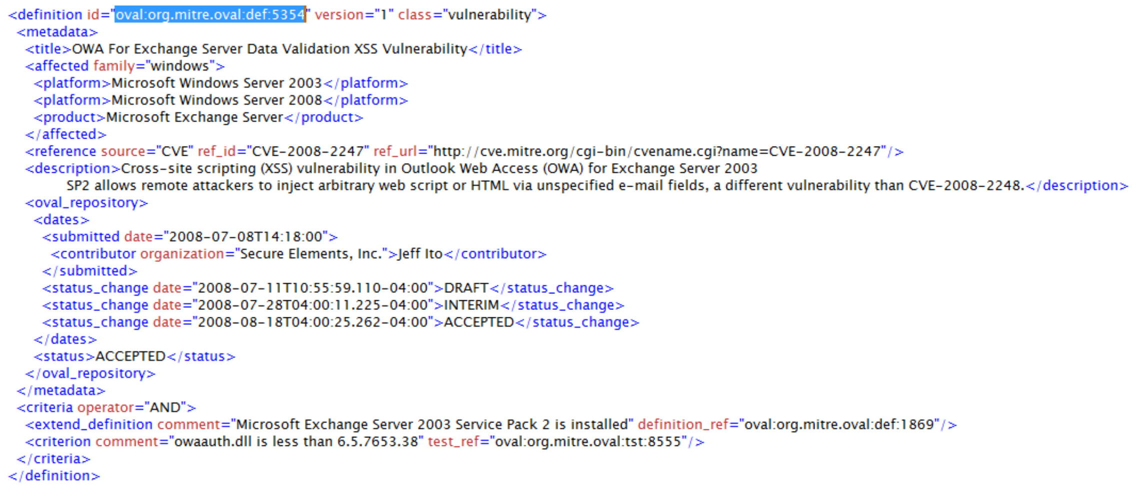


Figure 8: OVAL Definition Core Structure

In Figure 8, core schema has many components of the definition. The structure of the OVAL definition contains two main categories: metadata and criteria. Metadata includes information of each definition and refers to CVE. The description in metadata shows how this vulnerability could happen. The criteria in Figure 8 show how to draw this vulnerability by specifying which security check should be performed on the system. It has two categories: *extend_definition* mainly deals with the configuration of application, hardware, or operating system and *criterion* is to scan configurations by checking whether it meets any specific conditions.

To provide vulnerable information to different environments, we need to implement a flexible database which handles various structures for the target environments. Classical relational database or XML has limitations to provide such flexibility. To represent RDF triple mentioned in Figure 6, the classical relational database needs an additional table to link values and *join* operation to return data to the requester. The XML also needs many lines to represent these data and relationships in the system. With the help of ontology, this problem could be handled by using RDF triple. Taking advantage of this flexibility, we can share various vulnerability data with different environments.



```
<definition id="oval:org.mitre.oval:def:5354" version="1" class="vulnerability">
 <metadata>
  <title>OWA For Exchange Server Data Validation XSS Vulnerability</title>
  <affected family="windows">
   <platform>Microsoft Windows Server 2003</platform>
   <platform>Microsoft Windows Server 2008</platform>
   <product>Microsoft Exchange Server</product>
  </affected>
  <reference source="CVE" ref_id="CVE-2008-2247" ref_url="http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2247"/>
  <description>Cross-site scripting (XSS) vulnerability in Outlook Web Access (OWA) for Exchange Server 2003
        SP2 allows remote attackers to inject arbitrary web script or HTML via unspecified e-mail fields, a different vulnerability than CVE-2008-2248.</description>
  <oval_repository>
   <dates>
    <submitted date="2008-07-08T14:18:00">
     <contributor organization="Secure Elements, Inc.">Jeff Ito</contributor>
    </submitted>
    <status_change date="2008-07-11T10:55:59.110-04:00">DRAFT</status_change>
    <status_change date="2008-07-28T04:00:11.225-04:00">INTERIM</status_change>
    <status_change date="2008-08-18T04:00:25.262-04:00">ACCEPTED</status_change>
   </dates>
   <status>ACCEPTED</status>
  </oval_repository>
 </metadata>
 <criteria operator="AND">
  <extend_definition comment="Microsoft Exchange Server 2003 Service Pack 2 is installed" definition_ref="oval:org.mitre.oval:def:1869"/>
  <criterion comment="owaauth.dll is less than 6.5.7653.38" test_ref="oval:org.mitre.oval:tst:8555"/>
 </criteria>
</definition>
```

Figure 9, OVAL definition in XML

In Figure 9, core schema is described. For example, the title of this schema is OWA For Exchange Server Data Validation XSS Vulnerability. And its affected family is Microsoft Windows operating system and the reference shows the CVE identifier.

The component schema contains a specific path (object) and values (state) that identify the system configuration, which matches the vulnerability. Definition is composed of many different vulnerable configurations. Each criterion has its own test that contains an object and a state with a specific path and certain value, respectively. The matching of two paths and values declares a security issue but it may not a real vulnerability at this point. Extended definition is to check the installed software. The combination of the criterion and extended definition can finally declare a specific vulnerability.

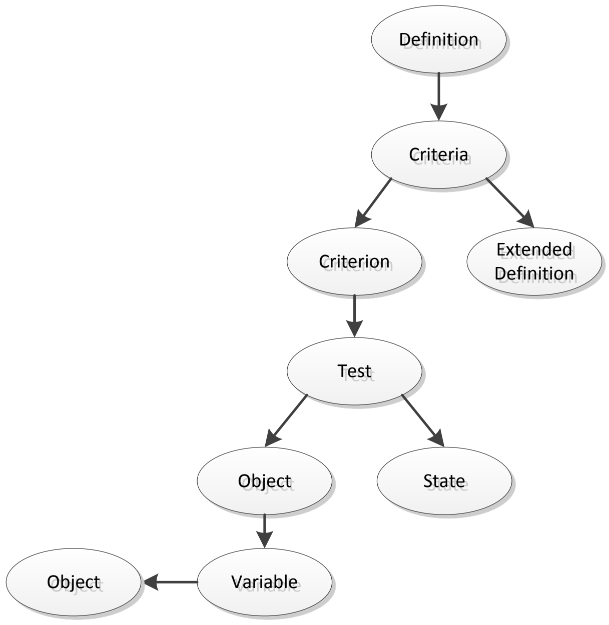
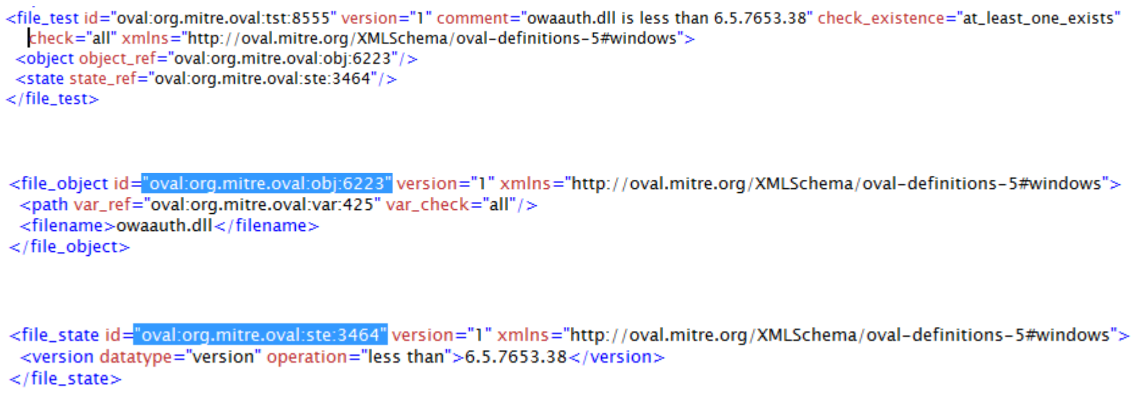


Figure 10: Criteria components schema of OVAL definition

In Figure 10, the structure of the component schema is illustrated. By checking object and state in a test, system administrators verify whether the test hits the vulnerable configuration on the system. In Figure 11, a specific example in this test has one object that shows the file ‘owaauth.dll’ and its state ‘6.5.7653.38’ with the path.



```
<file_test id="oval:org.mitre.oval:tst:8555" version="1" comment="owaauth.dll is less than 6.5.7653.38" check_existence="at_least_one_exists"
   check="all" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
  <object object_ref="oval:org.mitre.oval:obj:6223"/>
  <state state_ref="oval:org.mitre.oval:ste:3464"/>
</file_test>


<file_object id="oval:org.mitre.oval:obj:6223" version="1" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
  <path var_ref="oval:org.mitre.oval:var:425" var_check="all"/>
  <filename>owaauth.dll</filename>
</file_object>


<file_state id="oval:org.mitre.oval:ste:3464" version="1" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows">
  <version datatype="version" operation="less than">6.5.7653.38</version>
</file_state>
```

Figure 11: Criterion of test in OVAL XML

OVAL criteria have two operators: ‘and’ and ‘or’. The combination of ‘and’ and ‘or’ helps define a vulnerability in the OVAL definition. Criterion variable refers to another object, which shares the same path.

To use OVAL in many places, we design the basic structure of OVAL ontology to include attributes described in the OVAL definition, test, object and state related to the environments.

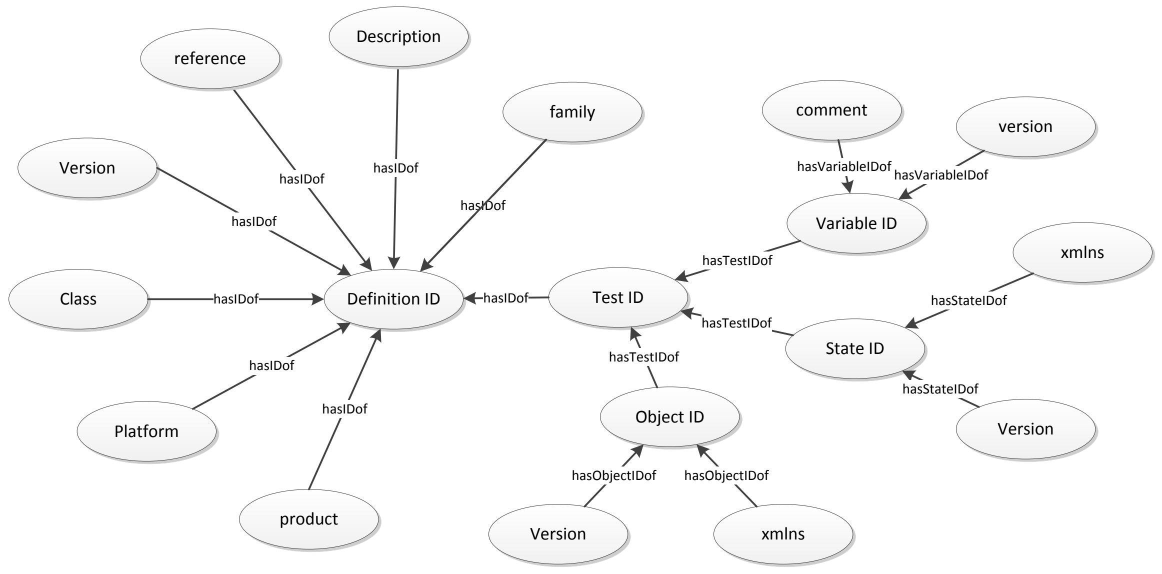


Figure 12: OVAL Ontology: Basic Structure

In Figure 12, we show the basic structure of the OVAL definition applied in ontology. *Definition ID* is an unique identifier and *Test ID* is the attribute which could be duplicated in different *Definition ID*s. So, connections between *Definition ID* and *Test ID* can be reusable in other relations. This basic structure is helpful not only understanding structure of OVAL ontology, but also further expanding information in different attributes. For example, the registry appears in only Microsoft Windows platforms. As mentioned previously, object has the path of the vulnerability so that a registry path information can be added to the *Object ID*. In the same way, the value of registry is added in the *State ID*. This expansion allows ontology to support diverse structures of OVAL definition and have the tool return its data for taking care of many different systems by using relationships in RDF triple. Also, ontology enables users to add any relationship into ontology without schema modification. In addition to such advantages, we attempt to improve our structure for enhancing the performance in returning information.

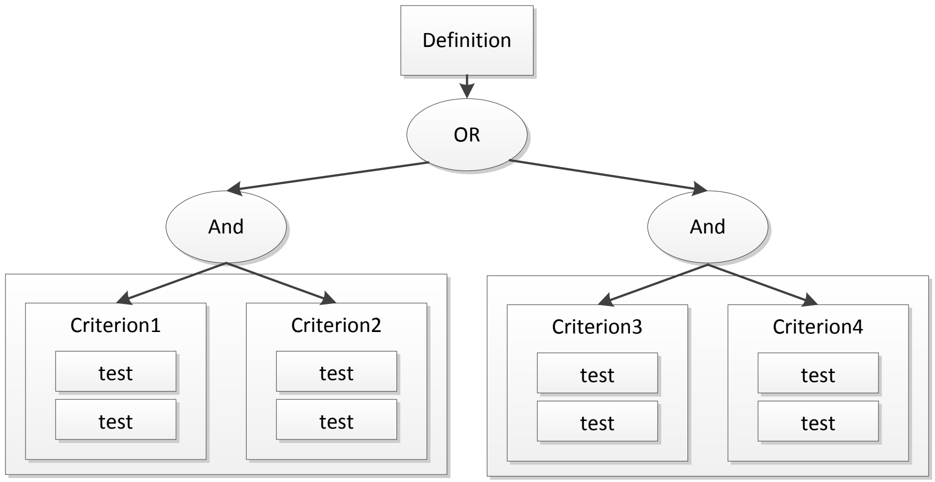


Figure 13: OVAL Criteria Operators

Every definition has its own criteria. Figure 13 shows a decision path based on logical operators to facilitate various criteria. For example, the definition indicates that it would be vulnerability if either criteria 1 and 2 or criteria 3 and 4 meet conditions mentioned in tests. In OVAL XML file, the published assessment tool should check all criteria until a criteria match conditions in the definition. Also, it is constructed as a hierarchical structure, which facilitates top-down approach. For instance, it would first check the version of operating system and product in metadata of a definition. Then it starts checking criteria until it finds matched combination of criteria. To overcome the performance issue of such a top-down but exhaustive approach, we introduce the notion of clustered area for checking the system effectively. The subsequent section will discuss our enhancement.

# 4. EVENT-DRIVEN CONTINUOUS MONITORING FRAMEWORK

The continuous monitoring has been recognized as a critical strategy and it could be realized by sharing incident information among government agencies and various organizations. As mentioned in previous chapters, this strategy has mainly focused on the way for continuously detecting and publishing new vulnerabilities or configuration problems. Figure 14 briefly describes the risk management framework proposed by NIST [16][17]. Based on this framework, this thesis concentrates on tasks in the phase 6: continuous monitoring. The life cycle of the framework determines whether the system meets the security requirements periodically but does not consider any changes with respect to the user's behavior such as installing applications or patching updates. Even though security assessment periodically generates and maintains vulnerability information in the security repository, ensuring system assurance and protecting target systems would be in vain without considering various vulnerability information and continuously monitoring configuration changes. Therefore, the system should perform event-driven comprehensive security assessment and environment independent monitoring.

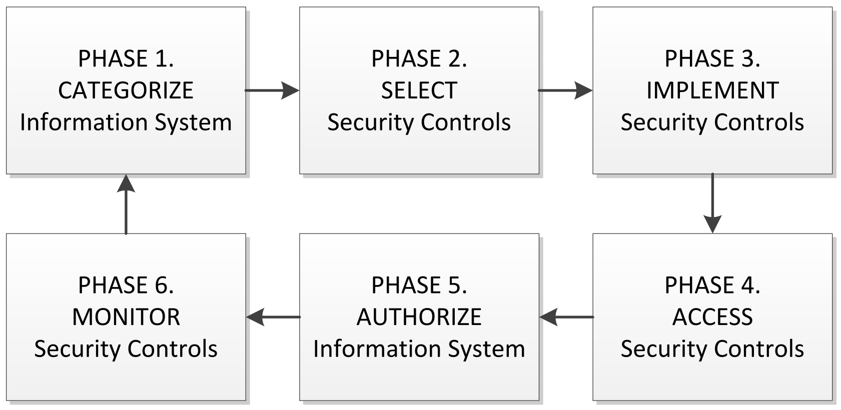


Figure 14: Risk Management Framework

Normally, the process of security monitoring deals with detection of risks in the system and active management of the detected risks. By adopting this method, security risks must be checked whenever suspicious events occur. In this thesis, we extend such conventional processes to include event-driven monitoring that facilitates usage reduction of system resources and event-related configuration check.

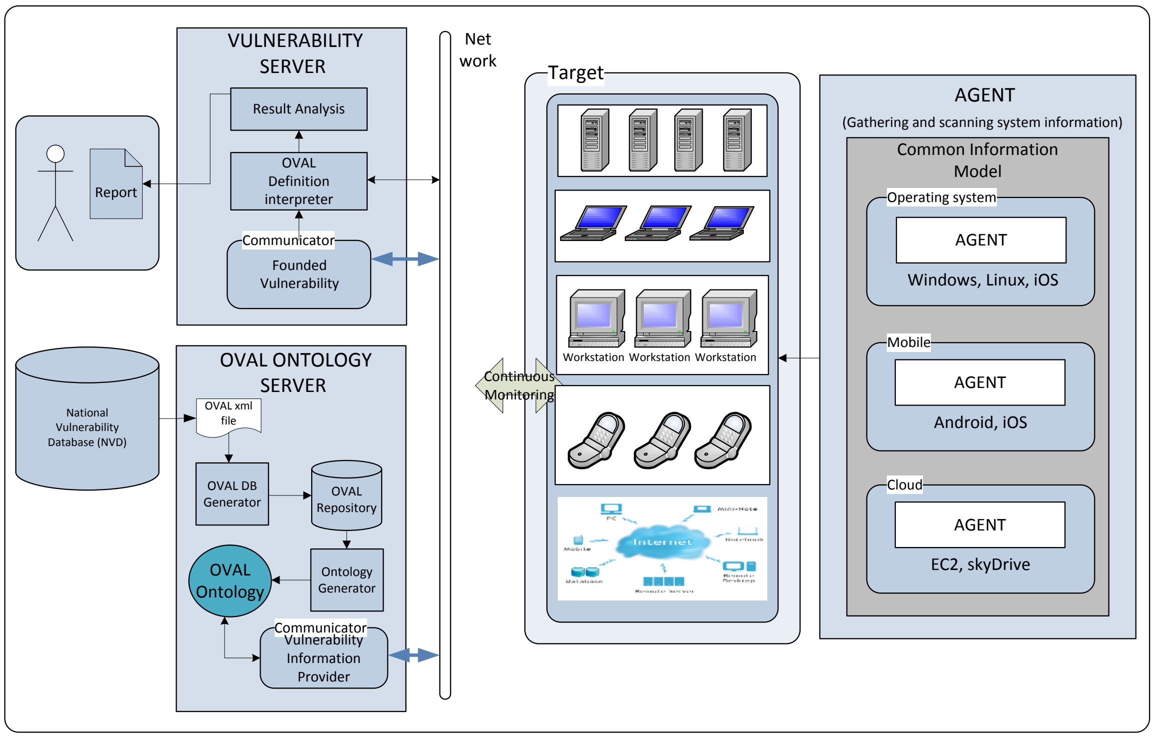


Figure 15: Event-driven continuous monitoring framework

## 4.1 GENERAL OVERVIEW

To provide comprehensive vulnerability information and environment independent event-driven continuous monitoring in the target system, we propose the event-driven continuous monitoring framework. The proposed framework has three main domains as is illustrated in Figure 15: vulnerability server, OVAL ontology server, and agent. Each domain is designed to perform specific tasks as follows: ontology server is to generate vulnerability information from OVAL fed by National Vulnerability Database (NVD) and provide security data to the agent through the network. To provide information that can support a target system in different environments, we also introduce high-level reasoning with vulnerability information to extract data based on environmental characteristics given by agents. Agents play an important role for gathering and scanning system information based on ontology. By using CIM discussed in the previous chapter, agents can be deployed in not only different operating systems, but also various devices such as mobile and cloud platforms. Agents can get notification of configuration changes by using CIM event classes. The role of vulnerability server is to search machines with the deployed agent in the network, receive detected vulnerabilities from agents, and verify if those vulnerabilities really exist in a target system. For analyzing and verifying the target system, we also use an OVAL interpreter to generate a report. OVAL interpreter validates the target system and generates the results for the security administrators.

## 4.2 EVENT-DRIVEN CONTINUOUS MONITORING

Minimizing computation costs and increasing assessment reliability of a target system are primary roles of the agent. We present an agent model that can detect security vulnerabilities in the system. Usually, vulnerability is considered as a logical combination of properties that can be presented in the target system. Properties in the system can vary depending on the nature of environments and security problems are associated with vulnerabilities in the system. One simple example of vulnerability is a specific running process (e.g. httpd), a specific open port (e.g. 80), and a specific version of the system (e.g. 2.6.10.rc). In other words, vulnerability may require several properties. To monitor events from a system, CIM event log which is one of CIM model for operating system information is used. WMI which is Microsoft extension of CIM is to support the CIM model for performing retrieval and event notification of the system. With such benefits of the system, data is accessed by COM/DCOM API in providers. There are many built-in providers in WMI. Among many providers, event providers handle event-driven continuous monitoring, which captures events and notifies the consumer. Windows NT event log provider provides access to data and event notification from the Windows NT event log [18]. When Windows NT is booted, it starts the Service Control Managers (SCMs). The win32 program event logging service is started up automatically when SCM started. Once an event is occurred in a device driver, or an application, it sends the report to the event logging service. The service stores the information that can be categorized as one of three event log files located in the local system disk: Application Event Log file, Security Event Log file, and System Event Log file.

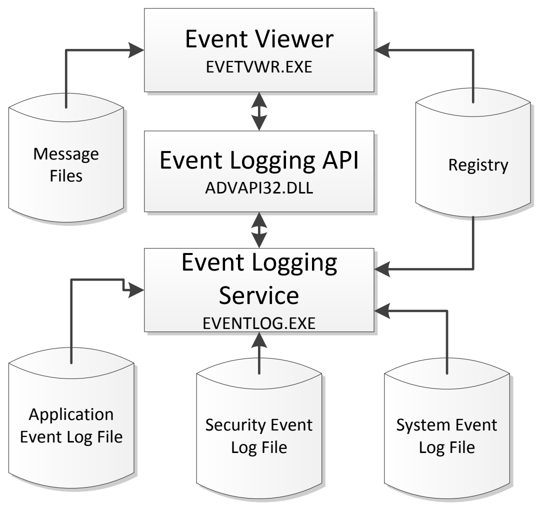


Figure 16: Event log service in Win32 program

WMI consumer in figure 3 can also retrieve a particular event from event repository for the further analysis. Figure 16 shows the structure of Win32 event logging service and how event logging service displays events from each log file. There exist two ways to access the event log files: local and remote. By using scripts or execution of a program that a system provides, event log file can be locally accessible. Also, the event logging service can be accessed by processes running on the local system. With remote procedure calls (RPCs), a remote computer can also access the event logs on the local system. All requests such as write, read, clear and backup operations on remote event logs are forwarded to the service using RPC. Both ways are transparent to the calling process. If WMI consumer requests and receives every event from log files checks security breaches related to the single event, automated security assessment requires heavy interactions with a running operating system to get system information and causes performance overhead. So, it is necessary to only extract data from related certain changes of a path described in security standards. This means that the system monitors and gathers data specified in the OVAL definition, instead of getting every event from the system. In this thesis, we narrow down the scope of this task to focus on Microsoft Windows operating systems. Every registry event is stored into Windows NT registry. The EventLog registry key is composed of event log sub-keys and event source keys. Event log sub-key stores the event log information for a specific registry event. Sub-key is mapped to the event source key. When an event source key is added to the registry, the name of the event source is automatically added to the source value of the corresponding event log sub-key by the event logging service.  Event logging service has functions to log the registry and file related events such as creating, modifying, and deleting the value of registry or file.

By using this event log service, changing information of certain file or registry can be monitored and used for the comparison between current and standard values.

CIM has an abstraction of event logger class for the system event log. In CIM, a CIM indication represents the occurrence of an event that changes the state of the environment of the component of the environment. For example, indicating the one service in the operating system is started or stopped or a certain application is installed so configuration is changed accordingly. An instance of the CIM_Indciation class represents the concrete indication of the occurrence of an event. Modeling CIM life cycle events of InstIndication includes instance creation, deletion, modification, method invocation and read access. From the security perspective, a system needs to distinguish interesting events from all collected events. This helps save resource usages and improves performance of security assessment by narrowing down the assessment scope of current system. For example, WMI provides WMI Registry Event Classes that can obtain registry information to interact with vulnerability information provider. Registry Event Classes is derived from _SystemClass class. The Registry event classes have four classes: RegistryEvent, RegistryKeyChangeEvent, RegistryTreeChangeEvent, and RegistryValueChangeEvent. RegistryEvent is an abstraction class for deriving other registry event classes. RegistryValueChangeEvent focuses on the changed value of a specific key since it can facilitate the specific path and value specified in the OVAL-based security standard. Other classes such as RegistryKeyChangeEvent and RegistryTreeChangeEvent monitor subkeys so additional process steps are needed. The indication of an event is made by Windows Query Language (WQL), which is the subset of ANSI Structured Query Language (SQL) in Microsoft Windows. The syntax of WQL is similar to SQL. WQL makes a system to get specific event information by narrowing the scope of an event. File changes in Microsoft Windows are also detected by using WMI. By comparing between changes and security standards, a system can determine whether changes on system configuration can affect the level of system security.

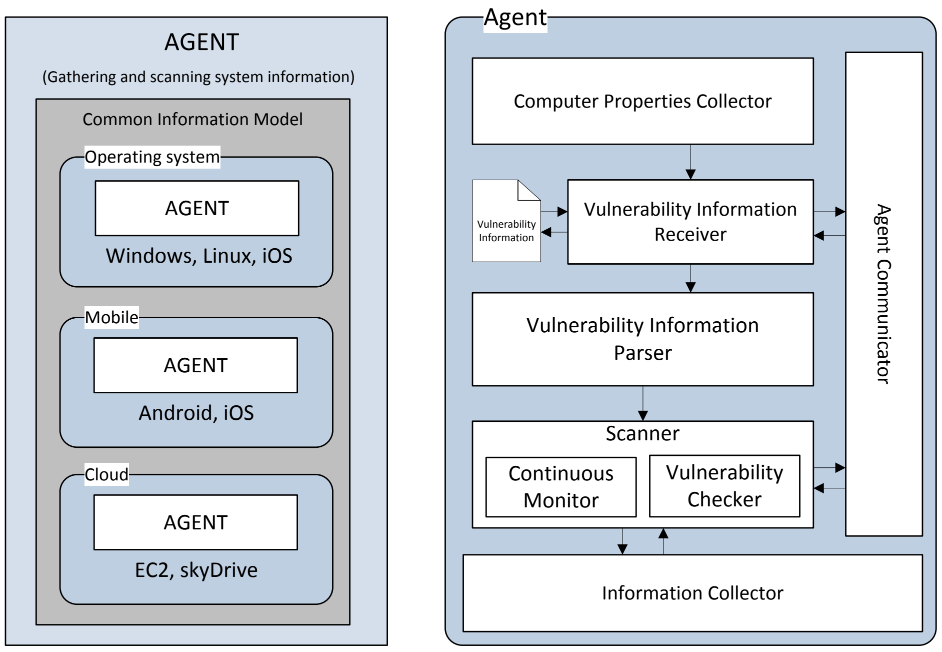


Figure 17: Agent Diagram

The agent architecture is depicted in Figure 17. To achieve continuous monitoring in various environments, the agent needs to monitor any events related to security risks and generate results when changes violate standard configuration. The agent is generic based on CIM object so that it can be used not only any operating systems, but also any other environments such as mobile and cloud as shown in Figure 17. The continuous monitoring in an agent starts with collecting system properties that is basis to get basic configuration and compliance information from standards. The agent sends such computer characteristics to the server and receives security vulnerability information to check security risks and monitor future changes with respect to the vulnerability described in the security standard. Based on collected computer information, ontology server extracts and provides configurations, compliances, and vulnerability information to the agent. The agent parses such information from ontology server and checks initial system state of a target system. This initial check of the target system leads the system to retrieve current vulnerabilities against standards and notify which vulnerabilities are found. After the initial check, the agent continuously monitors events associated with the vulnerability information received from ontology server. For configuration checking, the agent has different comparison methods because each operating system has its own structure. For example, Microsoft Windows operating system has registry as hierarchical database manager. Registry contains information how and what program is installed in while other operating systems store individual files in the file system. Scanner scans and checks each value based on security data received from OVAL ontology server. Continuous monitoring thread captures events from providers continuously and vulnerability checker thread compares captured events and standards for each event.

The agent detects vulnerability based on the following 4-tuples: (R, F, W, M), where R is the set of registry vulnerability; F is the set of file vulnerability; W is the set of WQL vulnerability; and M is the set of Metabase vulnerability of IIS configurations.

Consider R, F, M, and W has each subset as follows:

$$\mathrm{RN} = \{\mathrm{R}_1, \dots, \mathrm{R}_n\}, \text{where RN is the set of registry} \qquad (1)$$

$$\mathrm{FN} = \{\mathrm{F}_1, \dots, \mathrm{F}_n\}, \text{where FN is the set of file} \qquad (2)$$

$$\mathrm{MN} = \{\mathrm{M}_1, \dots, \mathrm{M}_n\}, \text{where MN is the set of Metabase} \qquad (3)$$

$$\mathrm{WN} = \{\mathrm{W}_1, \dots, \mathrm{W}_n\}, \text{where WN is the set of WMI} \qquad (4)$$

Each component is represented as the composition of path and value of each configuration:

$$R_i = \exists(Registry\ path, Registry\ value, \exists(partial\ order)) \qquad (1)\text{-}1$$

$$F_i = \exists(File\ path, Registry\ value, \exists(partial\ order)) \qquad (2)\ \text{-}1$$

$$M_i = \exists(Metabase\ path, Metabase\ value, \exists(partial\ order)) \qquad (3)\ \text{-}1$$

$$W_i = \exists(WMI\ Query, \exists(value), \exists(partial\ order)) \qquad (4)\text{-}1$$

WMI query checks WMI information with two elements: value and existence. Existence is to check the existence status of certain query described in $W_i$. From (1), (2), (3), and (4) definitions, the notion of vulnerability is formally defined as follows:

$$Vulnerability = \left\{\cup_{0\le i\le j\le q\le k\le n} \left(R_i \times F_j \times M_k \times W_l\right)\right\},$$

$$where\ R_i \in RN, F_j \in FN, M_k \in MN, W_l \in WN \qquad (5)$$

The following example shows how this definition can be realized with real world cases: Consider OVAL definition has a vulnerability description {*oval:org.mitre.oval:def:996*}which deals with file and print sharing service in Microsoft Windows operating systems. For instance, Microsoft Windows 95, Windows 98, and Windows ME do not properly check the password for a file share, which allows remote attackers to bypass access controls by sending a 1-byte password that matches the first character of the real password. This vulnerability is defined with the following composition of configuration properties: Vulnerability = $\{R_1 \times R_2 \times F_1\}$. This definition includes two registries and one file value. The first registry checks a key path of *HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion* which value is Windows 98. The other registry checks the existence of the following path *SYSTEM\CurrentControlSet\Service\UitlMan{5c773859-bb96-48fa-875b-6a58aae072f4}*. In addition, the value of the following file *%windir%\System\vserver.vxd* is checked to determine whether its value is less than 4.10.2001.0. Once these *R1*, *R2*, and *F* meet the each condition, the system declares a vulnerability entitled oval:org.mitre.oval:def:966 is detected. Information collector waits for an event which is triggered by any environmental changes. However, by processing only related information, our approach enhances performance by reducing the number of comparisons.

## 4.3 VULNERABILITY INFORMATION MANAGEMENT

To provide security standards to the agent in a seamless manner, we introduce ontology as a security information provider. The system has to handle various existing documents containing security problems published by different organizations. Even though each standard document has a structured format, its structures and attributes vary. Therefore, making a generic structure regarding to the security standards is needed. To address this issue, we use ontology because it provides not only data and its relationships, but also foundation of high-level reasoning. Particularly, a formal logic based on well-defined data and knowledge bases in ontology helps users deduce the implicit and inherent knowledge. In our system, ontology can extract and provide information about operating system vulnerability, application security issues, and security metrics as mentioned in OVAL. Based on preloaded data from ontology, agent can monitor and generate reports of current system status associated with vulnerabilities based on OVAL. To reduce errors to generate ontology from OVAL XML, the data structure is needed to verify each vulnerability definition since OVAL definition has different structures in criteria part.

The ontology approach consists of two parts: the first part is vulnerabilities based on the operating systems and its dependencies of operating systems and the second part is to get vulnerability with various structures. The second part of ontology approach needs two inputs: security event and its operating system.

The ontology is coded in OWL (Web Ontology Language [20]) and the Jena API which supports various types of OWL. The base case for algebra is a set of triple patterns which is called basic graph patterns (BGP). The example of BGP is as follows:

$$\{(?X, \text{name}, ?\text{name}), (?X, \text{email}, ?\text{email})\}$$

In this example, we can retrieve information of name and email of an entity X. A graph pattern expression from basic BGP is based on dom(μ) which is the domain of μ and μ(P) which is the set obtained from a basic graph pattern P. The example of BGP is as follows:

$$\mu = \{?X \rightarrow R1, ?\text{name} \rightarrow \text{john}, ?\text{email} \rightarrow \underline{\text{j@ed.ex}}\}$$

$$P = \{(?X, \text{name}, ?\text{name}), (?X, \text{email}, ?\text{email})\}$$

$$\mu(P) = \{(R1, \text{name}, \text{john}), (R1, \text{email}, \underline{\text{j@ed.ed)}}\}$$

The mapping μ1, and μ2 are compatible if and only if they agree in their shared variables: μ1(?X) = μ2(?X) for every ?X ∈ dom(μ1) ∪ dom (μ2).

$$\mu 1 = \{?X \rightarrow R1, ?name \rightarrow John\}$$

$$\mu 2 = \{?X \rightarrow R1, ?email \rightarrow \underline{j@ed.ed}\}$$

$$\mu 1 \cup \mu 2 = \{?X \rightarrow R1, ?name \rightarrow John, ?email \rightarrow \underline{J@ed.ex}\}$$

The evaluation of the BGP P over a graph G is denoted as $[[P]]_G$, which is the set of all mappings μ such that dom(μ) is the set of variables in P and μ(P) is the subset of G. The triple patterns and BGP are used for both system information extraction and detection of vulnerability based on OVAL language.

### 4.3.1 OPERATING SYSTEM BASED SECURITY TEST

Figure 18 and 20 show OVAL definition and criterion which are connected to test id. It also shows the extended definition that includes reference to the other definition in OVAL language. Metadata in OVAL definition contains many attributes and one of attributes is an affected attribute. An affected attribute has two sub attributes: platform and product and the further information are described in [21]. Platform has the operating system information that the definition can be applied to. By providing platform information in xml, the identified operating system will be checked by each definition. Product indicates that the definition is applicable when a specific application is installed in a target system. Breaking the OVAL language into components, a schema enables tools to reduce process overhead and execution time. However, the drawback of this approach is that schema structure is difficult for a user to navigate since its structure varies so it would have many different structures to be covered. Definition combines one or more tests using logical operator AND or OR operator. It wraps metadata and criteria to understand what and how processes will be taken for checking vulnerability on the target system. Definition has a unique id which starts with ‘oval:’ followed by three letter code ‘def’, and ending with integer. Each id is associated with criteria that outline what will be tested. The criteria consist of one or more tests with logical AND or OR operator.

Again, the process overhead and execution in continuous monitoring are important factors to achieve in our framework. In case that OVAL language continuously performs checking and validating tests and reference definitions, the system resources will be easily exhausted and execution time would be dramatically increased. Thus, our approach uses only environment related data via ontology. The basic idea is to enumerate all configurations including their values and potential vulnerable settings on a target system.

In Microsoft Windows operating systems, there are four types of vulnerability: File, registry, WQL, and Metabase. To get vulnerability information based on type, we provide a formal definition used in SPARQL for the extraction of the data from basic definition:

$$\left[\left[\{\left(\mu_1(P_1) OPT\ \mu_2(P_2)\right)\ FILTER\ \mu_3(P_3)\}\right]\right]_G$$

$$= \left\{\mu_k \in \left\{[[\mu_1(P_1)]]_G\ OPT\ [[\mu_2(P_2)]]_G\right\} \middle| \mu_k\ satisfies\ \mu_3(P_3)\ \right\} \qquad (6)$$

In the equation 6, the set $\mu_1(P_1)$ obtains vulnerability information based on operating system in a target system, while the set $\mu_2(P_2)$ handles null value which causes incorrect set of values. The set $\mu_3(P_3)$ categorizes the type of vulnerability returned from the test. This method shows vulnerability information is first extracted and then it is categorized by type based on the target system. The operator OPT is an extension of mappings in $M_1$ with compatible mappings in $M_2$ and FILTER returns the value only if it satisfies the $\mu$ defined in dom($\mu$).

### 4.3.2 DETECTION OF VULNERABILITIY

Guided by the SPARQL protocol, RDF query language and BGP in ontology framework, we harness the expressiveness of ontology to classify OVAL definition information from the following dimensions: 1) metadata including detailed information of vulnerability definition; 2) criteria containing regulatory checking method; and 3) domain-specific taxonomies of related test cases based on OVAL. Using object-oriented ontology into an interconnected definition, it can be easily expanded to address any domains.

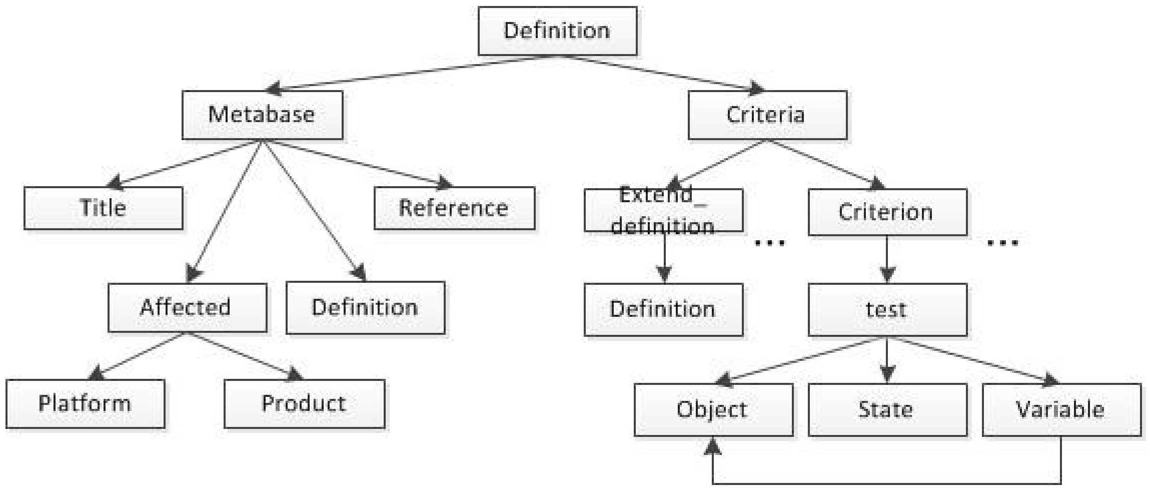


Figure 18: OVAL structure

In Figure 18, OVAL structure is described. The structure of OVAL varies based on operating system because each operating system has its own system. The part of metadata shows information related to definition like operating system product, which causes potential security issues. And criterion in a criteria part includes vulnerability path and value that will be used for checking each configuration. For example, when an event including installing application or updating patch occurs, agent tries to discover corresponding objects and states from standard. These objects and states compose one and more test ids so agent sends its test id to server for identifying additional information related to a single vulnerability definition. When server receives information from the agent, it starts to figure out additional information. Ontology for OVAL was derived from OVAL structure. However, OVAL structures vary so ontology is the best way to address the connection of anomalies among different operating systems.

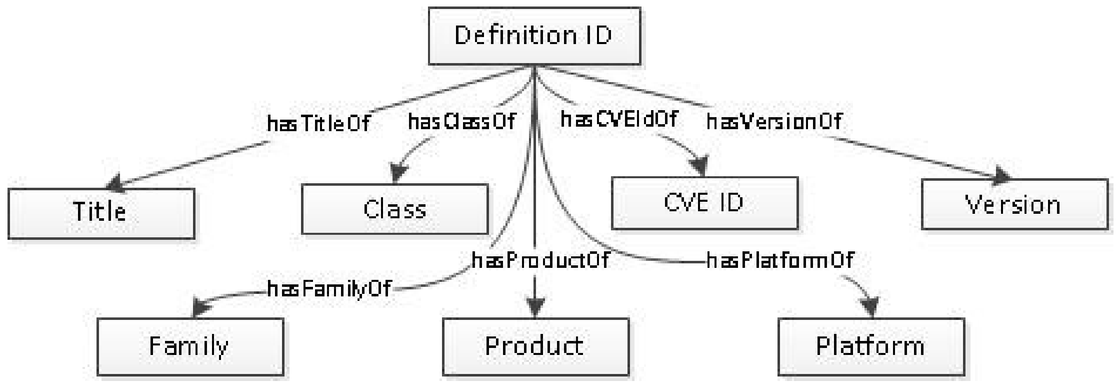


Figure 19: Metadata structure in ontology

Figure 19 shows how metadata structure for a definition is constructed based on OVAL. Each definition has a unique ID and the definition ID is straightly linked to attributes in metadata for providing accurate information. The direct link from the definition ID to its attributes can reduce unnecessary steps to reach the attributes that should be extracted. For example, the system will use CVE ID associated with detected vulnerability definition to calculate Common Vulnerability Scoring System (CVSS) scores. To calculate risk scores based on event, ontology provides additional test id which relates to vulnerability from detected events.

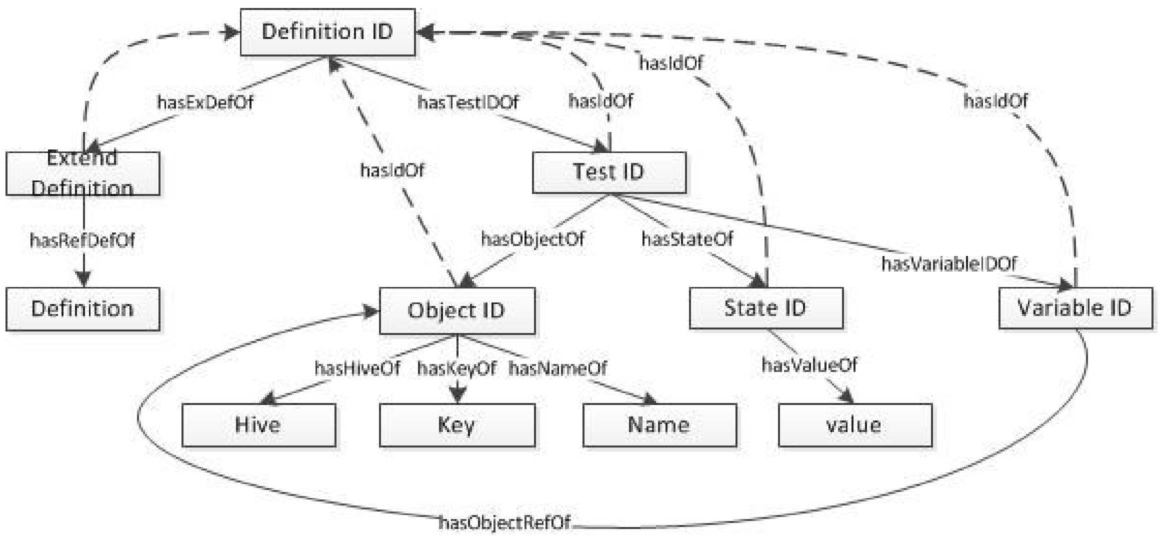


Figure 20: Criteria structure in ontology

As illustrated in Figure 20, the definition ID has at least one test id and one extended definition. Extend definition refers to another definition for verifying the configuration of the target computer such as application, operating system and hardware. Referenced definition has its own metadata and test id for the information. Definition can include more than one test id. As shown in Figure 13, the vulnerability can be detected by matching conditions of criteria. So, it is unnecessary to check all test ids. Instead, we introduce the notion of clustered test ids to overcome exhaustive check with the conditions in all test ids,

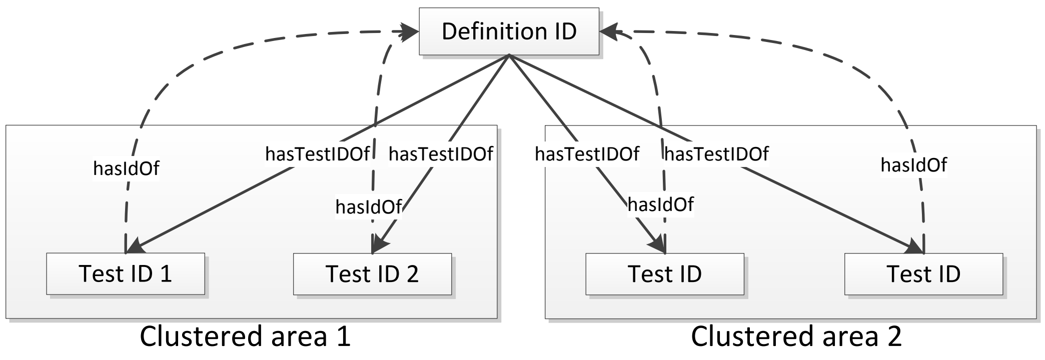


Figure 21: Clustered test ids

Clustered test id is constructed based on the type of operator. As shown in Figure 21, it implies that Test Ids 1 and 2 are clustered with ‘AND’ operator , while clustered areas 1 and 2 are associated with ‘OR’ operator.

## 4.4 SECURITY ASSESSMENT METHODOLOGY

As mentioned earlier, NVD [22] provides not only standardized information of most software products available today, but also risk level of each product by score associated with CVE included in OVAL. The score is from CVSS [23] which is the tool that enables security administrator to quantify the severity and risk of individual product. However, CVSS could not be used directly to measure vulnerability in a particular product because the design of CVSS only aims at an individual vulnerability. Due to this reason, it is sometimes ignored that one product installed in a target system could cause multiple vulnerabilities in the system. Moreover, most risk assessment approaches based on CVSS do not reflect the concerns from security administrator who performs and demands such security assessment tasks. The risk level can vary based on different viewpoints on the assessment results. For example, if a security administrator is more concerned with the impact on the system, the assessment results should give more weight on the impact of a particular vulnerability. To achieve this goal, we use CPE, CVE and CVSS to gather risk information assess the security posture of the software with the given weight provided by security administrators. In the subsequent chapters, we present our approach to obtain vulnerability measured by OVAL and store vulnerability information into our repository. Then, we analyze the collected information using CPE, CVE and CVSS based on exploitability and impact aspects.

### 4.4.1 MEASUREMENT OF SECURITY RISK

Rigorous and continuing risk assessment substantially helps protect systems from risks and threats. If we use CVSS scoring system to measure risks in the system, we have to deal with many CVSS scores since one product may have multiple vulnerabilities and the vulnerability has many CVSS score based on status of product. It is necessary to consider all vulnerabilities for calculating a risk score based on CVSS scores. Moreover, the importance of computer is varied depending on the purpose of each system so that security administrators may need to perform security assessment along with their own security concern.

The CVSS base score metrics contain six vectors: Access Vector (AV), Access Complexity (AC), Authentication (AU), Confidentiality Impact (CI), Integrity Impact (II), and Availability Impact (AI). Three factors, AV, AC, and AU show current state of exploit techniques or code availability. It measures availability of exploit codes that could increase the attack vector when it is available. For example, suppose web browser has a set of ‘network’ in AV, ‘none’ in AU, and ‘low’ in AC. It describes that vulnerability exists in a web browser if network is accessible, no authentication is required, and this sever is easy to access. The impact of the system indicates how much it could be compromised by the identified vulnerability.

When OVAL generates the results in XML format, Common Platform Enumeration (CPE) is included for defining and explaining the conformance of IT products [34]. CPE has three categories: operating system, application, and hardware. In our approach, we check the system including running applications, operating system, and hardware with OVAL and categorize the detected vulnerability scores by CPE.

As we mentioned above, we categorize CVE based on CPE. Suppose a software product is $p$ in CPE and the number of vulnerability is $n$ associated with the product $p$ which is denoted as follows:

$$p = \sum_{i=1}^{n} pi$$

All CVEs in a product $p$ is $T$ which is represented as follows:

$$T = AV \times AC \times AI \times II \times CI \times AI$$

We first separate the CVSS scores into two groups. The one of groups is categorized by exploitability as follows:

In $T \ni EX_i$ and $T \ni EX_j$

$, EX_i = \{AV_i, AC_i, AU_i, II_i, CI_i, AI_i\}$ *and* $EX_j = \{AV_j, AC_j, AU_j, II_j, CI_j, AI_j\}$

$EX_i \approx EX_j$ *where* $AV_i = AV_j$, $AC_i = AC_j$, *and* $AU_i = AU_j$

This means that a group is categorized only if the scores of AV, AC, and AU are identical while, II, CI, and AI do not need to be same. The exploitability-based risk score is represented as follows:

$$EX^{ex} = \sum_{i=1}^{n} (EX_k)_i$$

The set of impact $IM_k$ corresponding to the $EX^{ex}$ is $IM^{ex}$, the impact-based risk score is formulated as follows:

$$IM^{ex} = \sum_{j=1}^{n} (IM_k)_j$$

In addition, we calculate the vulnerability of IT product based on weights from the security administrators. Each weight shows the importance of concern on each vector: $w_{a,}$ $w_{\beta}$ and $w_{\gamma}$ denotes the importance of operating system (*o*), importance of hardware (*h*) and the importance of application (*a*), respectively. Also, we introduce two additional inputs from security administrators to express their concerns between exploitability and impact factors: $w_{ex}$ represents the exploitability weight and $w_{im}$ shows the impact weight. By using these weights and vulnerability scores grouped by CPE, we can compute the overall vulnerability scores.

Exploitability-based scoring method is represented as follows:

$$p_i(os_i) = \left(w_\alpha * o + w_\beta * h + w_\gamma * a\right)$$
$$* \left\{\sum_{i=1}^{n}\left(\left(\frac{EX_i^{ex}}{\sum_{j=1}^{n} EX_j^{ex}}\right) * w_{ex}\right) + \sum_{i=1}^{m}\left(\left(\frac{IM_i^{ex}}{\sum_{j=1}^{m} IM_j^{ex}}\right) * w_{im}\right)\right\}$$

The assessment categorized by impact is formulated as follows:

$$p_i(os_i) = (w_\alpha * o + w_\beta * h + w_\gamma * a) * \left\{ \sum_{i=1}^{n} \left( \left( \frac{EX_i^{im}}{\sum_{j=1}^{n} EX_j^{im}} \right) * w_{ex} \right) + \sum_{i=1}^{m} \left( \left( \frac{IM_i^{im}}{\sum_{j=1}^{m} IM_j^{im}} \right) * w_{im} \right) \right\}$$

If the vulnerability is categorized by exploitability, the same exploitability will be added to the group without considering impact values.

As shown in the formula above, we define the risk level based on CPE and CVSS by the given weights from security administrator. Given information in the server, we first analyze the installed IT products by CPE and identify possible combination of exploitability (*AV, AC, AU*) and impact (*II, CI, AI*) values. These combinations can be basic characteristics of the vulnerability and it reflects IT product characteristics as well. The combination of exploitability and impact can be 27 possible cases, respectively. So, there exist 54 cases as security metrics that we can use to evaluate. For each case, we can also rank vulnerability among others based on security administrator's point of view.

# 5. IMPLEMENTATION AND EVALUATION

In order to realize the proposed approaches in Chapter 4, we implement the automatic system to measure security vulnerability in a target system and generate risk analysis results. In our implementation, we focus on the Microsoft Windows operating systems

As mentioned in Figure 14, the architecture consists of three components: OVAL ontology server, vulnerability server, and agent. OVAL ontology server is the server providing vulnerability information to the agent in conjunction with the target computer's characteristics and it also returns relevant vulnerabilities. Vulnerability server verifies certain vulnerability and invokes risk assessment in a target system. The role of agent is to provide system characteristics and monitor potential fault configurations that can be exploited by the attacker. In subsequent chapters, we will discuss how we implemented each component and evaluation results.

## 5.1 IMPLEMENTATION DETAILS

Our framework is realized as an event-based monitoring system. Figure 22 shows a high level architecture of our system with three components. The left part of architecture is ontology server that can support information to the agent which runs in various environments. The agent exists in the middle to provide continuous monitoring vulnerable events that may cause vulnerability in the system. And the rightmost component in the architecture is the vulnerability server, which is used by security administrator to figure out the level of risk for a target system.

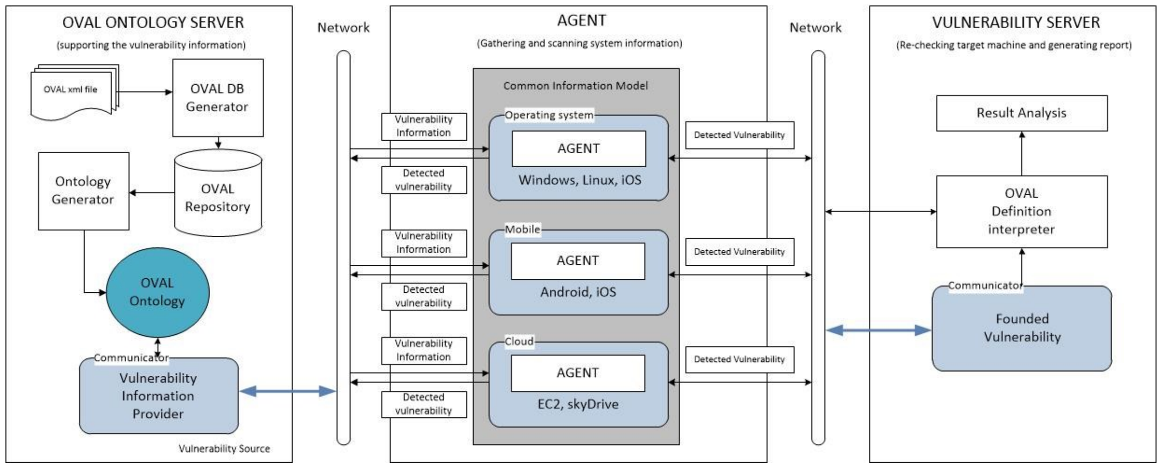


Figure 22: Event-driven Continuous Monitoring Architecture

In this chapter, we first discuss implementation details of each component in the event-driven continuous monitoring framework. Then, we articulate the features of our security assessment.

### 5.1.1 AGENT

Agent modules are implemented in Java and j-Interop library which enable systems to interoperate with COM and DCOM components. The agent delivers computer characteristic of the installed machine, receives the server information from vulnerability server, finds matched vulnerability on the current configuration and monitors events relating to the ones specified in the standards. The agent has functionalities to gather system characteristics and send characteristics to vulnerability server. Once vulnerability server picks an agent in a target system, vulnerability server sends server information so that the agent can collect vulnerable configurations from ontology server. The vulnerability information receiver module passes the received security configuration information to parse through configuration information with the regular expression.

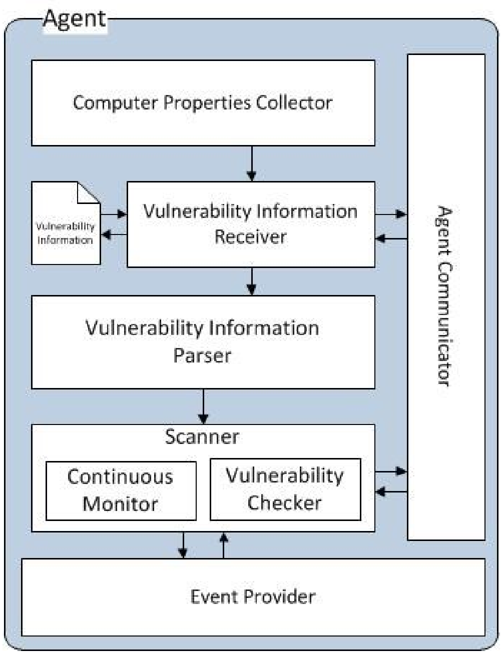

Agent
Computer Properties Collector
Vulnerability Information
Vulnerability Information Receiver
Vulnerability Information Parser
Scanner
Continuous Monitor
Vulnerability Checker
Agent Communicator
Event Provider

Figure 23: Agent Diagram

The scanner component consisting of two sub modules: continuous monitor and vulnerability checker in Figure 23. The vulnerability checker is to scan vulnerability information from vulnerability information parser and figure out existing security problems on the system. The continuous monitor is mainly focused on the event which can trigger security issues on the system. The continuous monitor subscribes specific paths described in security standards in OVAL ontology server for detecting vulnerable events caused by end users. Once the vulnerable events are occurred and detected by the continuous monitor, the Subhandler in scanner requests additional information to the server with respect to the matched events. Vulnerability checker re-scans additional information in the system and determines current changes related to security breaches on the system. And detected vulnerabilities in the target system are sent to the vulnerability server by Subhandler.

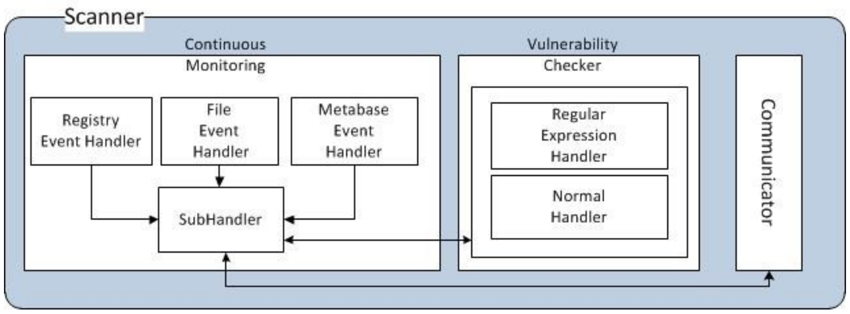


Figure 24: Continuous Monitor Diagram

Continuous monitor consists of three threads to check multiple events at the same time. Security standards for Microsoft Windows operating systems mainly deal with four configurations: registry, file, Metabase and WQL. Event handler monitors the changes and reports vulnerable configurations. Of these event handlers, WQL is not event-driven so it does not need to be monitored. Three event monitors keep watching the changes of system configuration continuously. Each event handler receives the notification of an event from WMI repository. One of handlers is the *RegistryEventHandler* which waits for an event related to the registry path and value. This handler uses *RegistryTreeChangeEvent* class in WMI class which observes a path and its sub-path in the registry by using two conditions: hive and root_path of a specific path. *RegistryTreeChangeEvent* can capture three events: creation, modification, and deletion of sub-registry path or value. However, this *RegistryTreeChangeEvent* class does not monitor a non-existent path so that pre-creation is required before starting to monitor. Another handler is the *FileEventHanlder* which gathers the collection of data from WMI by using *__InstanceOperationEvent*. This *__InstanceOperationEvent* class monitors particular files, including files that do not exist in the logical drive currently. The

*__InstanceOperationEvent* consists of three classes: *__InstanceCreationEvent*, *__InstanceDeletionEvent*, and *__InstanceModificationEvent*. Each class stores events of creation, deletion, and modification of specific information on a file, respectively. The *CIM_DATAFILE* class represents the collection of data related to a target file. By using these two classes, scanner can get which and how file is changed by an event. The last handler is *MetabaseEventHandler* which uses *CIM_DATAFILE* to look at Metabase file which is the collection of data for Internet Information Service (IIS). As the handler uses *CIM_DATAFILE*, it can detect changes on file modification of the Metabase file in the system. The continuous monitoring function has *SubHandler* which handles sub-procedures when an event is considered as vulnerability captured by each *EventHandler*. *SubHandler* requests the event detected by *SubHandler* and checks additional vulnerability information to confirm if security problems occur. *SubHandler* has its own thread until finishing the task so that the agent could have many *SubHandlers* depending on the number of events. *SubHandler* has different functions comparing with *EventHandler* that handles regular expressions. Since OVAL definition provides a path or a value of the vulnerability information based on a regular expression so that *SubHandler* needs to manage regular expression in a proper manner to capture fault configurations in the system.

**Algorithm 1** Intitial Checker

```
1: procedure INITIAL CHECKER
2:     $character \leftarrow$ collectSystemCharacteristics()
3:     $dataList \leftarrow$ getVulnerabilityInfoFromServer($character$)
4:     $foundList \leftarrow$ Checker($dataList$)
5:     subChecker($foundList$)
6: end procedure
```

Algorithm 1 shows how an initial check works without a regular expression. Initial checker compares security standards with system configuration since the target computer might have potential security problems in the current system configuration. The initial checker starts with the collection of system specifications. When it gathers system characteristics, it sends the collected data to ontology server and gets vulnerability information. Vulnerability information is then used to check the target system. As summarized in Algorithm 2, the checker receives data list from the initial checker and takes types to check vulnerabilities associated with the type of vulnerability. As mentioned earlier, Microsoft Windows operating systems have four types of vulnerability determined by the configuration check. The tool scans configurations and compares it with the vulnerability information from the previous procedure in Algorithm 1. The agent sends the detected vulnerability information to vulnerability server and vulnerability server generates a report based on the initial check.

**Algorithm 2** Checking Computer System

```
procedure CHECKER($dataList$)
    while $dataList$ is not null do
        $type \leftarrow$ getVulnerabilityType ($data$)
        if $type = registry$ then
            $found \leftarrow$ registryCheck($data$)
        else if $type = file$ then
            $found \leftarrow$ fileCheck($data$)
        else if $type = metabase$ then
            $found \leftarrow$ metabaseCheck($data$)
        else
            $found \leftarrow$ wqlCheck($data$)
        end if
        if $found$ eq $true$ then
            $foundList \leftarrow data$
        end if
    end while
    return $foundList$
end procedure
```

As discussed earlier, we use both normal expression and regular expression to compare the results from the initial configuration check. The algorithm 4 shows how to check a target system in regular expression. The regular expression handler continues to check if the registry path exists. The registry paths that are matched with regular expression will be stored and re-scanned to discover all sub-paths. If there is no certain path comparing with security standards, the path will be removed from the path list. After collecting all paths matched with this regular expression, the agent starts checking values in the target system.

**Algorithm 3** Regular Expression Handler

```
 1: procedure REGEXHANDLER($data$)
 2:     insert $data$ substring into $datapath$[]
 3:     $i \leftarrow 0$
 4:     while $i = (datapath\ array\ length)$ do
 5:         for $k = 0$ to $(paths\ array\ length)$ do
 6:             $currentpath \leftarrow parths[k]$
 7:         end for
 8:         $totalpath \leftarrow currentpath + datapath[i]$
 9:         if $datapath[i]$ is exist then
10:             $paths[i]$ = datapath[i]
11:         else
12:             remove $paths[i]$
13:             $i \leftarrow i$ - 1
14:             break;
15:         end if
16:         $i \leftarrow i + 1$
17:     end while
18:     if $i = (datapath\ array\ length)$ then
19:         $value \leftarrow$ get information by data path
20:         if $value$ = data value then
21:             vulnerability found
22:         else
23:             vulnerability not found
24:         end if
25:     end if
26: end procedure
```

### 5.1.2 ONTOLOGY SERVER

Ontology server was implemented by JAVA, MySQL, Apache JENA and SPARQL. Apache JENA is widely used ontology builder which is Java framework for semantic web applications. Ontology server uses OVAL definitions but data extraction procedure is different from the methods used by OVAL. To utilize security standards for a target system based on system characteristics received from the agent, we first transformed and stored OVAL definitions into ontology server so that we can build a well-defined knowledge base based on OVAL.

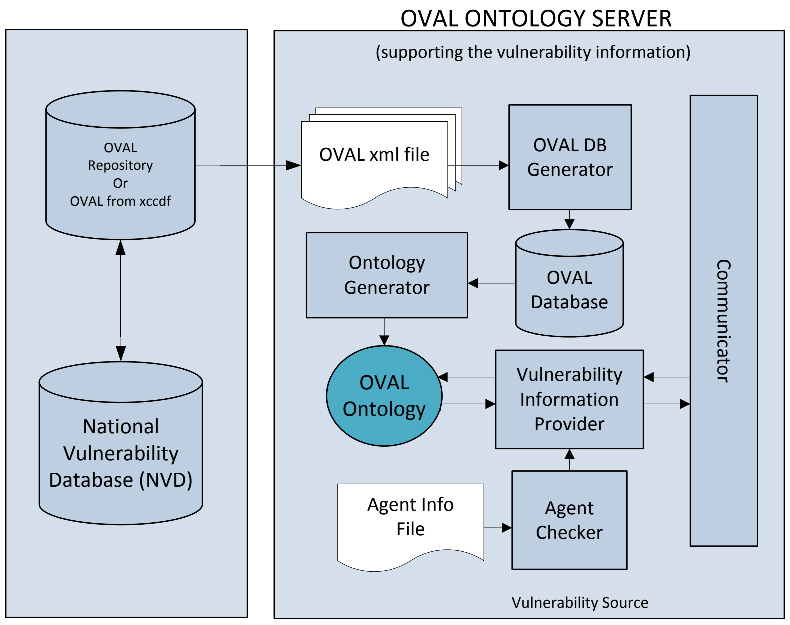


Figure 25: Ontology Server Diagram

Ontology server has several components for providing information to the agent. The OVAL DB generator parses and stores OVAL to database. The database is able to produce XML files that include the OVAL definitions. Criteria contained in OVAL definitions have different structures so there is possibility to occur errors while transforming ontology to OVAL XML files directly. Therefore, we leverage database to maintain OVAL information and reduce potential errors in the transformation [7]. As shown in Figure 25, DB generator parses OVAL XML file and stores it into database.

Since there are many criteria that contain vulnerability information by object and state, several tables can be created in database. In other words, one definition may have multiple criteria for defining vulnerability.

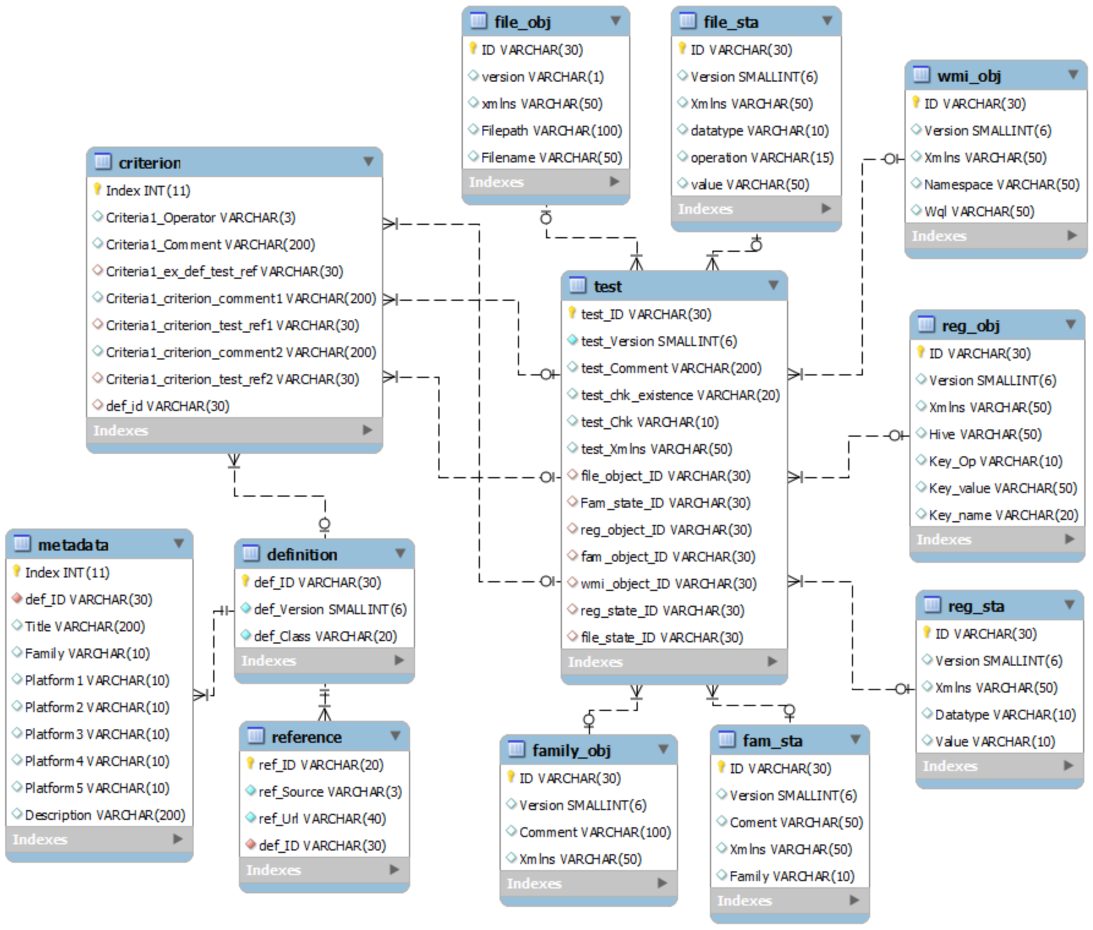
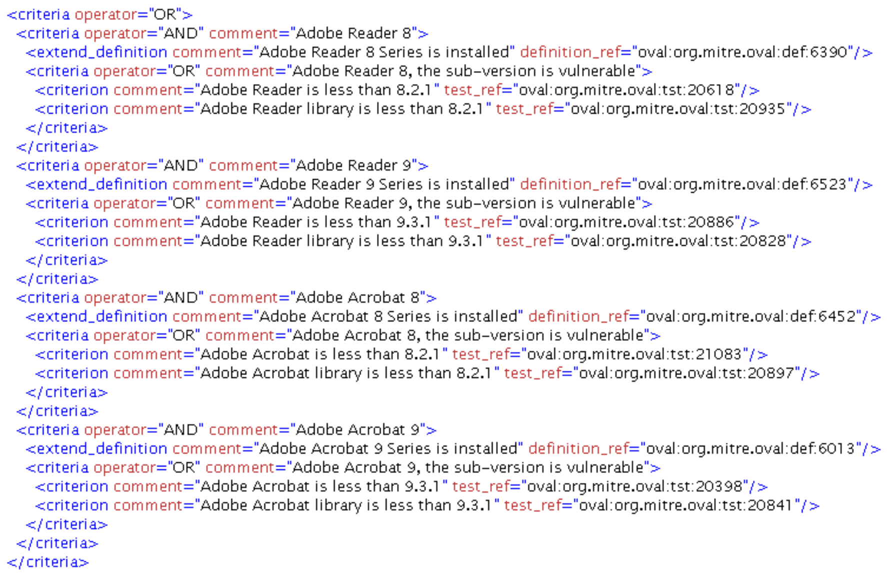


Figure 26: OVAL Repository database structure

To retrieve vulnerability information from database, the *join* operation should be committed. However, *join* operation combines tuples from different relations so it is relatively expensive operation [28]. Instead of optimizing join operation, we use ontology generator to create ontology file. The ontology generator retrieves data from database and generates ontology file by using Apache JENA library. By using the notion of clustered area, the server efficiently extracts data related to detected vulnerability in the system. In Chapter 5.2, we provided the detailed information how the ontology is constructed and handles the relevant tasks. The other component in ontology server is vulnerability information provider. The vulnerability information provider uses SPARQL to extract data from owl file that contains security data and its relation. This component has two functionalities to support the agent. One is to deliver security standards to the agent based on system characteristics gathered by the agent in a target system. The characteristics include MAC address, operating system, IP addresses, user id, password, and so on. The other is to supply test-based vulnerability information in the OVAL definition to the agent.



```
<criteria operator="OR">
 <criteria operator="AND" comment="Adobe Reader 8">
  <extend_definition comment="Adobe Reader 8 Series is installed" definition_ref="oval:org.mitre.oval:def:6390"/>
  <criteria operator="OR" comment="Adobe Reader 8, the sub-version is vulnerable">
   <criterion comment="Adobe Reader is less than 8.2.1" test_ref="oval:org.mitre.oval:tst:20618"/>
   <criterion comment="Adobe Reader library is less than 8.2.1" test_ref="oval:org.mitre.oval:tst:20935"/>
  </criteria>
 </criteria>
 <criteria operator="AND" comment="Adobe Reader 9">
  <extend_definition comment="Adobe Reader 9 Series is installed" definition_ref="oval:org.mitre.oval:def:6523"/>
  <criteria operator="OR" comment="Adobe Reader 9, the sub-version is vulnerable">
   <criterion comment="Adobe Reader is less than 9.3.1" test_ref="oval:org.mitre.oval:tst:20886"/>
   <criterion comment="Adobe Reader library is less than 9.3.1" test_ref="oval:org.mitre.oval:tst:20828"/>
  </criteria>
 </criteria>
 <criteria operator="AND" comment="Adobe Acrobat 8">
  <extend_definition comment="Adobe Acrobat 8 Series is installed" definition_ref="oval:org.mitre.oval:def:6452"/>
  <criteria operator="OR" comment="Adobe Acrobat 8, the sub-version is vulnerable">
   <criterion comment="Adobe Acrobat is less than 8.2.1" test_ref="oval:org.mitre.oval:tst:21083"/>
   <criterion comment="Adobe Acrobat library is less than 8.2.1" test_ref="oval:org.mitre.oval:tst:20897"/>
  </criteria>
 </criteria>
 <criteria operator="AND" comment="Adobe Acrobat 9">
  <extend_definition comment="Adobe Acrobat 9 Series is installed" definition_ref="oval:org.mitre.oval:def:6013"/>
  <criteria operator="OR" comment="Adobe Acrobat 9, the sub-version is vulnerable">
   <criterion comment="Adobe Acrobat is less than 9.3.1" test_ref="oval:org.mitre.oval:tst:20398"/>
   <criterion comment="Adobe Acrobat library is less than 9.3.1" test_ref="oval:org.mitre.oval:tst:20841"/>
  </criteria>
 </criteria>
</criteria>
```

Figure 27: OVAL definition criteria

Figure 27 shows a sample OVAL definition. This OVAL definition has four different criteria that cause security issues in Adobe Reader. If the agent detects *oval:org.mitre.oval:def:6390* in the system and then ontology server returns two test IDs such as *oval:org.mitre.oval:tst:20618* and *oval:org.mitre.oval:tst:20935* for helping the agent check the related vulnerability in the system.

### 5.1.3 VULNERABILITY SERVER

Vulnerability Server was implemented in Java. Based on the retrieved vulnerability information, this server re-verifies vulnerabilities that are detected by the agent in the target system. This server has three modules: result analysis, auto assessment, and agent management. Result analysis module helps security administrators generate a report based on the detected vulnerabilities. This module contains two components: security assessment and report display. Security assessment shows significant vulnerabilities in a target system based on the importance factor described in Chapter 4.4. Report display component provides analysis results of the target system. The second module is auto assessment that has three components: oval parser, oval merger, and verification tool. Oval parser is to split oval definitions by ID and oval merger merges only related IDs to scan the target system. The part of agent management mainly controls each agent and this module allows each agent to access both vulnerability server and ontology server. In other words, only authorized agents can access both servers.

The interpreter in auto assessment module only receives the merged oval information related to events acquired from agents. Since the oval definition interpreter needs information from a target system, the property handler helps the interpreter establish a session with a target system remotely by creating *config.properties*. Once auto assessment module receives vulnerability data from the agent, auto assessment module generates the vulnerability result through result analysis module including vulnerability information from document and the level of risk based on weights from security administrators.

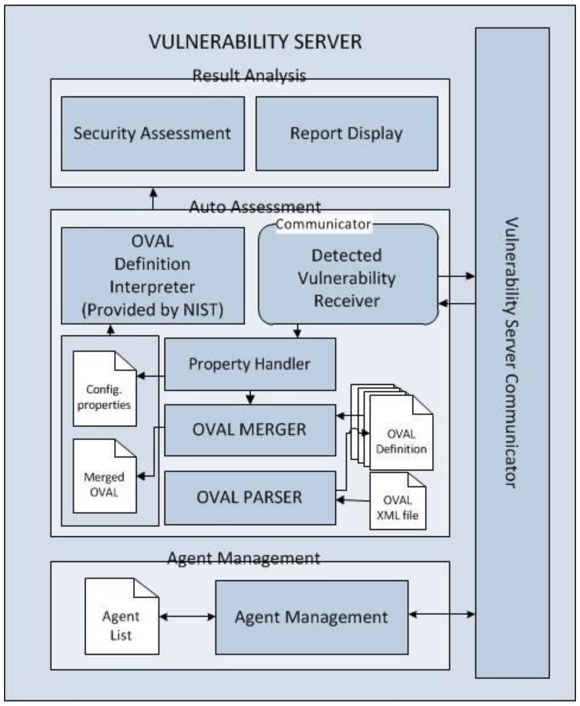


Figure 28: Vulnerability Server Architecture

Figure 28 depicts the above-mentioned procedures and Algorithm 4 summarizes our risk calculation approach mentioned in Chapter 4.4.1.

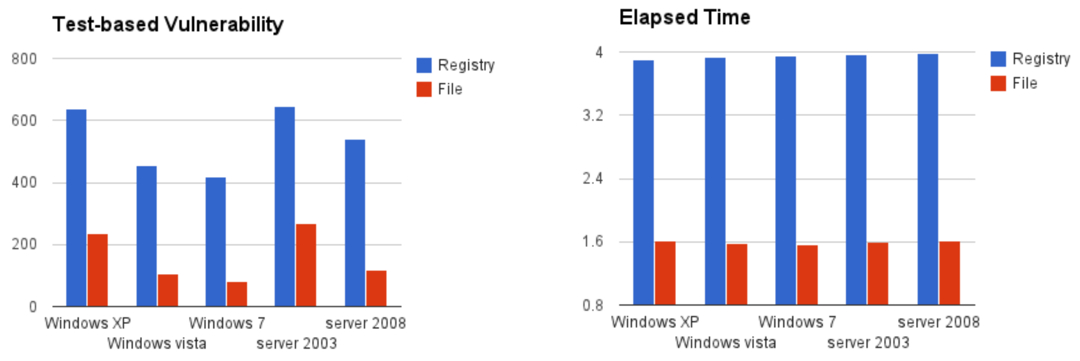
**Algorithm 4** Risk Assessment

```
procedure RISK CALCULATION($os$, $app$, $hw$, $ex$, $im$, $list$)
    maxIm = 0
    maxEx = 0
    for 0 to $cpearray.length$ do
        for 0 to $category.length$ do
            impact = 0
            exploitability = 0
            for 0 to $cve.length$ do
                impact ← impact + $cve.impactScore$
                exploitability ← exploitability + $cve.exploitability$
            end for
            category.setIm(impact)
            category.setEx(exploitability)
            maxIm ← maxIm + impact
            maxEx ← maxEx + exploitability
        end for
    end for
    for 0 to $cpeArray.length$ do
        result = 0
        for 0 to $cpeArray.category.length$ do
            result ← ((category.getEx()/maxEx) * $ex$) + ((category.getIm()/maxIm) * $im$)
            category.setScore($result$)
        end for
    end for
end procedure
```

## 5.2 EVALUATION RESULTS

In this chapter, we describe comprehensive and analytical evaluation results of our system to demonstrate the feasibility and scalability of our approach.

In order to test the effectiveness our solution, we measured the extraction time to retrieve vulnerability information from ontology. Our experiment was performed with a desktop computer (Core2 quad q9650 3.0 GHz CPU, 16GB RAM), and multiple Microsoft Windows operating systems including Windows XP, Windows Vista, Windows 7, Server 2003 and Server 2008. The extraction task is divided into two parts. One part is to measure the number of test-based vulnerability retrieved for a particular operating system. As mentioned earlier, our tool should check and obtain the path of the vulnerability before starting to launch a monitoring task. Therefore, this measurement shows whether our system legitimately retrieves relevant vulnerability.

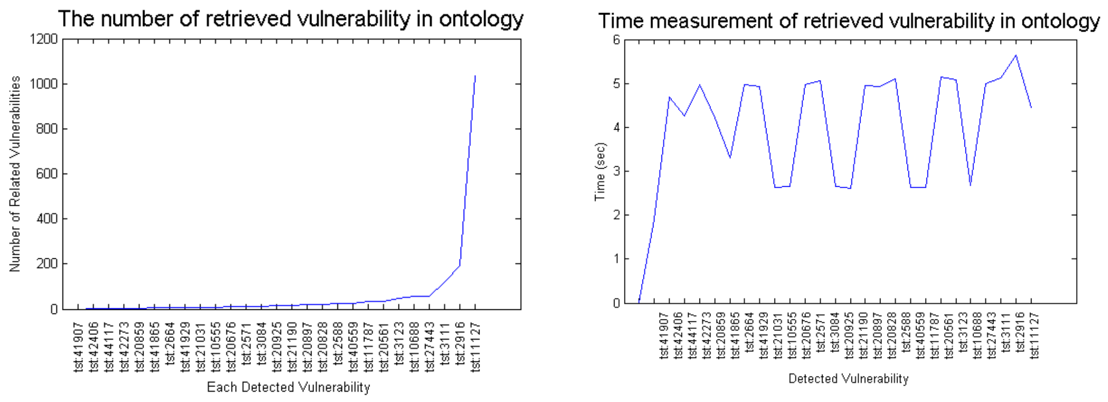


(1) Number of retrieved vulnerability (2) Vulnerability retrieval time

Figure 29: Performance measurement in ontology

Figure 29 shows the number of vulnerability retrieved from OVAL ontology and the performance in extracting vulnerability information based on registry and file from ontology.

The other part is to retrieve tests and definitions from OVAL ontology server. Figure 30 shows that the extraction time is consistent--no matter how many test ids are retrieved from ontology. In ontology, each definition contains two types of criterion: test and extended definition. The test contains paths and values for the vulnerability while the extended definition mainly refers to the other definition.



(1) Number of Retrieved Vulnerabilities (2) Respond Time of the Vulnerability

Figure 30: Performance measurement in ontology

Figure 30 (1) shows the number of vulnerabilities retrieved from ontology and (2) describes the extraction time of retrieved vulnerabilities regarding to the detected vulnerability in the agent. In Figure 30 (1), there are more than 1,000 vulnerabilities extracted by one test id occurred in target machine. We found two reasons why one test id could have a relation with many test ids. One reason is that there are many criterions which have a AND relationship with detected test id in the definition. And the other reason is that test id is used in the multiple definitions. In figure 30 (2), we analyze the different reasons for time variations: detection test id in clustered area and refer to different definition. The case of detection which is made in clustered area in Figure 21 decreases the extraction time. Based on analyses from both graphs, we derive the result that extraction time takes at most 5 seconds in many different numbers of retrieved vulnerabilities.

For the measurement of performance in agents, we built a testbed in a cloud by using Openstack. Each virtual machine image represents one of the following systems: Windows 7 and Windows Server 2008 respectively. For brevity, our measurement ignored network latency but focused on the vulnerability assessment.

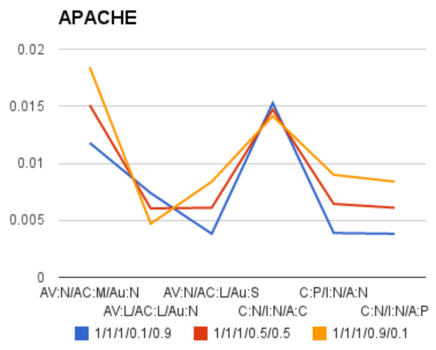
| Windows 7 | | | | |
|---|---|---|---|---|
| Envrionment | Number Of Registry | Registry Check Time (sec) | Number Of File | File Check Time (sec) |
| Core 1, 2GB RAM | 319 | 24.314 | 81 | 12.388 |
| Core 2, 2GB RAM | 319 | 14.887 | 81 | 15.536 |
| Core 4, 2GB RAM | 319 | 17.064 | 81 | 7.254 |

| Windows Server 2008 R2 | | | | |
|---|---|---|---|---|
| Envrionment | Number Of Registry | Registry Check Time (sec) | Number Of File | File Check Time (sec) |
| Core 1, 2GB RAM | 77 | 2.728 | 3 | 0.141 |
| Core 2, 2GB RAM | 77 | 2.979 | 3 | 0.062 |
| Core 4, 2GB RAM | 77 | 3.308 | 3 | 0.078 |

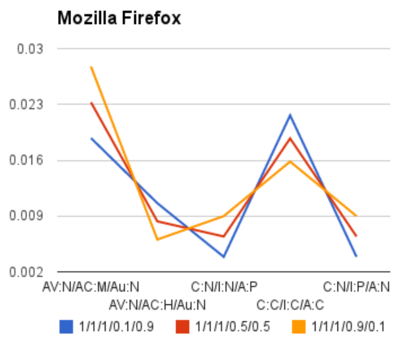
Table 1: Initial Evaluation in Agent Tool

As summarized in Table 1, the performance in single core and 2GB RAM takes 24 seconds to check the system but takes 15 seconds to scan and detect the vulnerability in the system. However, the results between dual core and 4GB RAM and quad core and 8GB RAM did not indicate any significant changes. In other words, our agent could perform the tasks in a timely manner without producing any unexpected overhead.

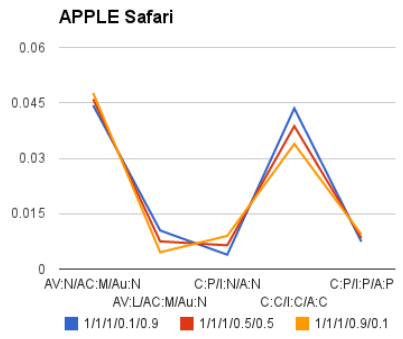
Based on the risk assessment approach introduced in Chapter 4.4.1, we have performed several experiments to determine how the weights from security administrators can affect our assessment results. The assessment considers at most 54 groups of vulnerability in a product found in a target system and each group is categorized by either exploitability or impact. The dataset is collected from XML files by jOVALdi tool. For instance, our tool checks vulnerability with a locally installed product in a target system and tested it again in vulnerability server with jOVALdi. In order to analyze results with given weights, we performed several experiments in a desktop computer (CPU 2.80GHz, 4.00GB RAM, Microsoft Windows 7 operating system).



(1) Assessment Result in Apache



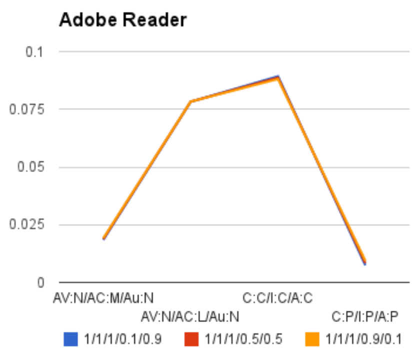
(2) Assessment Result in Firefox

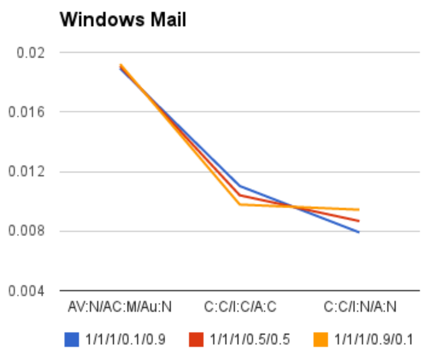


(3) Assessment Result in Safari

Figure 31: Assessment Results

Figure 31 illustrates the analysis results based on the different viewpoints from security administrator. All CPE values set to 1 because we mainly concentrate on changes in CVSS score based on given weights from security administrator. Figure 3 (1) shows that Apache product has vulnerabilities in the system but the most critical vulnerability in the system has been changed depending on the value of exploitability and impact. For example, the significant vulnerability was changed after setting up the exploitability impact to 0.9 and 0.1, respectively. We could obtain similar results in Mozilla Firefox and Apple Safari products. In addition, the result shows that these weights not only change the most significant vulnerability but also affect the ranking of vulnerabilities. As a matter of fact, none of critical vulnerability is affected.





(1) Assessment Results in Adobe Reader (2) Assessment Results in Windows

Figure 32: Assessment Results (Less Change)

We have observed that some experiments show almost constant results under different weights. Figure 32 illustrates both products were not affected by weights from security administrator. However, CPE factors such as operating system, application, and hardware, could change the significant vulnerability and ranking of vulnerability

# 6 CONCLUSION

In this thesis, we have proposed an innovative security assessment system that is designed to facilitate not only event–driven continuous monitoring, but also automated risk assessment accommodating various environmental requirements. Event-driven continuous monitoring system is capable of monitoring suspicious events, which could lead security risks based on security standards in OVAL. Also, the proposed system can be easily adapted to various environments in a seamless manner. In addition to the event-driven continuous monitoring, we have also introduced the tool that can provide and expand vulnerability information with high-level reasoning and decision-making. Our experiments demonstrated we could accomplish the comprehensive security risk assessment based on security administrator's view point.

## 6.1 CONTRIBUTION

The major contributions of this thesis are summarized as follows:

1. We articulated the need for event-driven continuous monitoring including identified challenges and design criteria in building corresponding security assessment systems.
2. We proposed systematic approaches to realize event-driven continuous monitoring framework that automatically assesses vulnerabilities and calculate risk scores based on multiple viewpoints from security administrators.
3. We implemented a proof-of-concept prototype based on our event-driven continuous monitoring framework. We evaluated our system with various use cases for each component and our results showed an event-driven continuous

monitoring system could analyze vulnerable configurations and calculate risk scores in a seamless and timely manner.

## 6.2 FUTURE WORK

Our future work includes the refinement and extensions of the event-driven continuous monitoring framework. The current approach cannot perform continuous monitoring for the registry events after the reinstallation of product followed by the uninstallation. This problem is attributed by one of followings: inability to create trace logs, stopped agent, and data corruption [30]. The inability to create trace logs is the main reason that subscriber of events cannot receive the event notification from the provider. We will further study to overcome this issue so that we can even detect any changes in system configurations caused by the uninstallation of products. In addition, our system is focused on Microsoft Windows operating systems. However, we can extend the system to support the different operating systems or environments in a seamless manner. We plan to extend our approach to support various environments based on CIM-based approach. In addition, we will investigate a more efficient and effective way to enhance an ontology-based security assessment including streaming reasoning and robust knowledge base for vulnerabilities.

# 7 REFERENCES

[1] Mirko Montanari, Roy H. Campbell, "Multi-aspects security configuration assessment", SafeConfig, November 9, 2009

[2] John Pescatore, "Dealing with Federal Continuous Monitoring Security Requirements", NIST SP 800-137, October 2012

[3] Stephen Quinn, Karen Scarfone, David Waltermire, "Guide to Adopting and Using the Security Content Automation Protocol (SCAP) Version 1.0", NIST SP 800-117, January 2012

[4] David Waltermire, Stephen Quinn, Karen Scarfone, Adam Halbardire, "The Technical Specification for the Security Content Automation Protocol (SCAP)", NIST SP 800-126, September 2011

[5] Ron Ross, "The Future of Cyber Security, NIST Special Publication 800-53, Revision 4", June 2013

[6] Guntars Bumans, "Mapping between Relational Databases and OWL Ontologies: an Example", Scientific papers, University of Latvia, Vol. 756, 2010

[7] MAN LI, XIAO-YONG DU, SHAN WANG, "Learning ontology from relational database", Machine Learning and Cybernetics, August 2005

[8] Department of Homeland Security Federal Network Security Branch, "Continuous Asset Evaluation, Situational Awareness, and Risk Scoring Reference Architecture Report (CAESARS)", September 2010

[9] Distributed Management Task Force (DMTF), "Profile for CIM", August 2009

[10] Jos de Bruijin, "Using Ontologies (enabling knowledge sharing and reuse on the semantic web)", DERI-2003, October 2003

[11] Ora Lassila, Ralph R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification", W3C, February 1999

[12] Dan Brickley, R.V. Guha, "RDF Vocabulary Description Language 1.0: RDF schema", W3C, February 2004

[13] Eric Prud'hommeaux, Andy SeaBorne, "SPARQL Query Language for RDF", W3C, January 2008

[14] Jiewen Huang, Daniel J. Abadi, Kun Ren, “Scalable SPARQL Querying of Large RDF Graphs”, VLDB 2011

[15] Stephen D. Quinn, Murugiah Souppaya, Melanie Cook, Karen Scarfone, “National Checklist Program for IT products – guidelines for checklist users and developers”, NIST SP 800-70, February 2011

[16] National Institute of Standards and Technology (NIST), “Guide for applying the risk management framework to federal information system”, February 2010

[17] National Institute of Standards and Technology (NIST), “Guide for assessing the security controls in federal information systems and organizations”, June 2010

[18] James Murray D. “Windows NT Event Logging 1st Edition”, September 1998

[19] http://www.informationweek.com/government/security/federal-cybersecurity-incidents-rocket-6/231700231

[20] Deborah L. McGuinness, Frank van Harmelen, “OWL web ontology language”, http://www.w3.org/TR/owl-features/, February 2004

[21] W3Counter for Global Web Stats, www.w3counter.com/globalstats.php, September 2013

[22] NHS and NIST, National Vulnerability Database (NVD), “automating vulnerabilities management, security measurement, and compliance checking”, http://nvd.nist.gov/scap.cfm

[23] Peter Mell, Karen Scarfon, and Sasha Romanosky, “A complete Guide to the Common Vulnerability Scoring System (CVSS)”, Version 2.0, Forum of Incident Response and Security Teams, http://www.first.org/cvss/cvss-guide.html

[24] JuAn Wang, Minzhe Guo, Hao Wang, “Ontology-based Security Assessment for Software Products”, CSIIRW, April 2009

[25] Bill TSOUMAS, Dimitris GRITZALIS, “Towards an Ontology-based Security Management”, AINA, 2006

[26] Dinesh Chandra Verma, “Principles of Computer Systems and Network Management 2009 edition”, 2009

[27] Edgar R. Weippi, “Improving Storage Concepts for Semantic Models and Ontologies”, Idea Group, 2009

[28] Carlos Ordonez, "Evaluating Join Performance on Relational Database Systems", Journal of Computing Science and Engineering, December 2010

[29] J. Patrick Thompson, "Web-Based Enterprise Management Architecture", IEEE Communications Magazine, March 1998

[30] IBM, "IBM Trivoli", Monitoring, Version 6.1, September 2006

[31] Fang Yu, Zhifeng Chen and Yanlei Diao, "Fast and memory-efficient regular expression matching for deep packet inspection", Architecture for Networking and Communications Systems, 2006

[32] Mark Dowd, John McDonald, Justin Schuh, "The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities 1st edition", Nov 2006

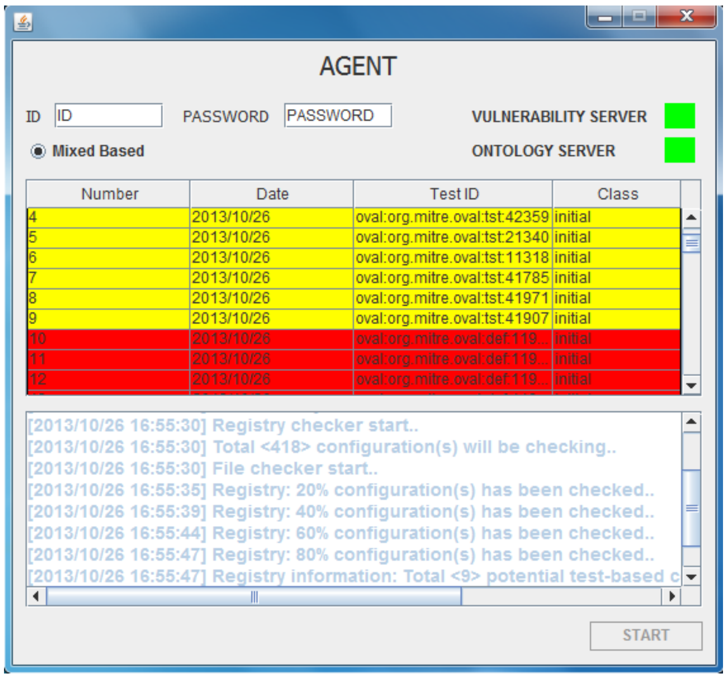
[33] National Institute of Standards and Technology (NIST), "Federal Desktop Core Configuration (FDCC)", http://fdcc.nist.gov/

[34] MITRE corporation, "Common Platform Enumeration (CPE)", http://cpe.mitre.org/

[35] Anreas Ekelhart, Stefan Fenz, Markus Klemen and Edgar Weippl, "Security Ontologies: Improving Quantitative Risk Analysis", HICSS, 2007

APPENDIX

- AGENT



ID: ID    PASSWORD: PASSWORD

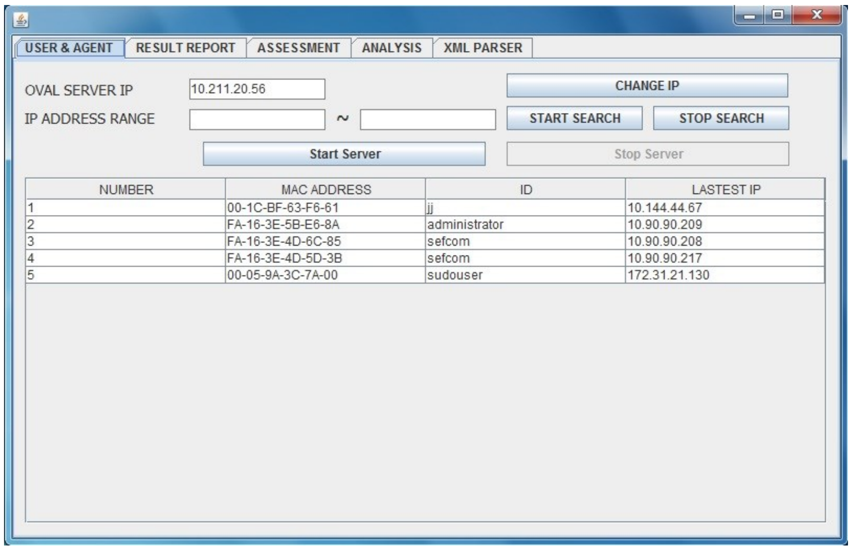
(●) Mixed Based

| Number | Date | Test ID | Class |
|---|---|---|---|
| 4 | 2013/10/26 | oval:org.mitre.oval:tst:42359 | initial |
| 5 | 2013/10/26 | oval:org.mitre.oval:tst:21340 | initial |
| 6 | 2013/10/26 | oval:org.mitre.oval:tst:11318 | initial |
| 7 | 2013/10/26 | oval:org.mitre.oval:tst:41785 | initial |
| 8 | 2013/10/26 | oval:org.mitre.oval:tst:41971 | initial |
| 9 | 2013/10/26 | oval:org.mitre.oval:tst:41907 | initial |
| 10 | 2013/10/26 | oval:org.mitre.oval:def:119... | initial |
| 11 | 2013/10/26 | oval:org.mitre.oval:def:119... | initial |
| 12 | 2013/10/26 | oval:org.mitre.oval:def:119... | initial |

```
[2013/10/26 16:55:30] Registry checker start..
[2013/10/26 16:55:30] Total <418> configuration(s) will be checking..
[2013/10/26 16:55:30] File checker start..
[2013/10/26 16:55:35] Registry: 20% configuration(s) has been checked..
[2013/10/26 16:55:39] Registry: 40% configuration(s) has been checked..
[2013/10/26 16:55:44] Registry: 60% configuration(s) has been checked..
[2013/10/26 16:55:47] Registry: 80% configuration(s) has been checked..
[2013/10/26 16:55:47] Registry information: Total <9> potential test-based c
```

1) ID and password to get information of both system and configuration
2) Agent console displaying the specific procedure for checking configuration of the machine

- Vulnerability Server
  - User and Agent Panel



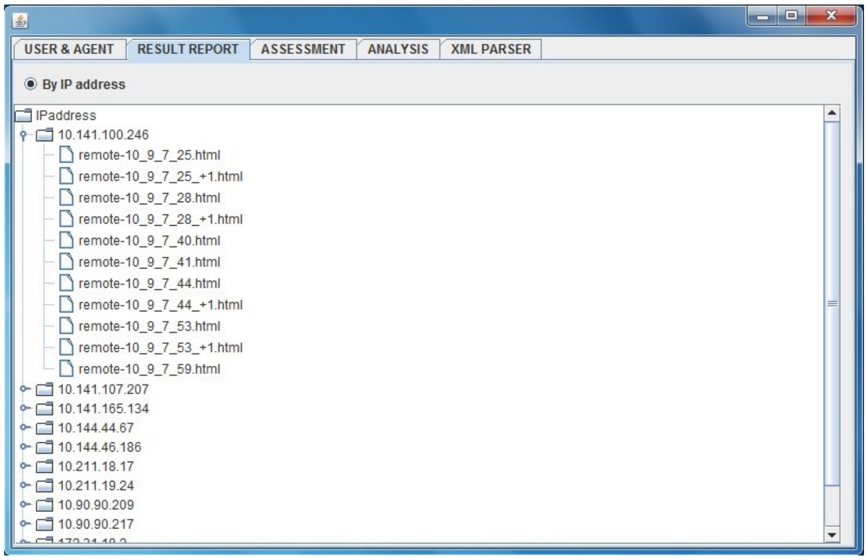
OVAL SERVER IP: 10.211.20.56

IP ADDRESS RANGE: ~

| NUMBER | MAC ADDRESS | ID | LASTEST IP |
|---|---|---|---|
| 1 | 00-1C-BF-63-F6-61 | jj | 10.144.44.67 |
| 2 | FA-16-3E-5B-E6-8A | administrater | 10.90.90.209 |
| 3 | FA-16-3E-4D-6C-85 | sefcom | 10.90.90.208 |
| 4 | FA-16-3E-4D-5D-3B | sefcom | 10.90.90.217 |
| 5 | 00-05-9A-3C-7A-00 | sudouser | 172.31.21.130 |

1) OVAL Server IP for agent connection
2) IP range to search agent installed system
3) Information table displaying target system information

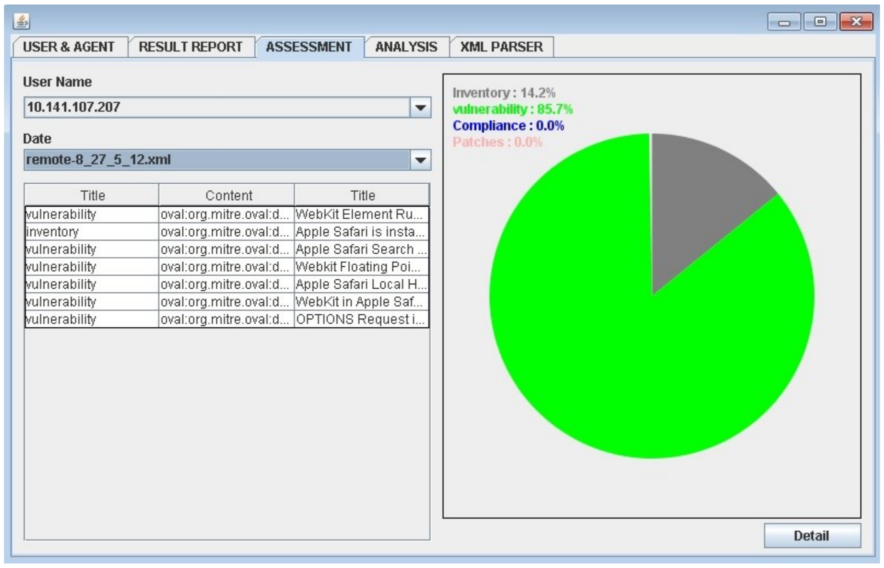
- Result Report Panel



```
IPaddress
  10.141.100.246
    remote-10_9_7_25.html
    remote-10_9_7_25_+1.html
    remote-10_9_7_28.html
    remote-10_9_7_28_+1.html
    remote-10_9_7_40.html
    remote-10_9_7_41.html
    remote-10_9_7_44.html
    remote-10_9_7_44_+1.html
    remote-10_9_7_53.html
    remote-10_9_7_53_+1.html
    remote-10_9_7_59.html
  10.141.107.207
  10.141.165.134
  10.144.44.67
  10.144.46.186
  10.211.18.17
  10.211.19.24
  10.90.90.209
  10.90.90.217
```

1) Reported result classified by IP address of the target system
2) Report display by double click

- Assessment Panel



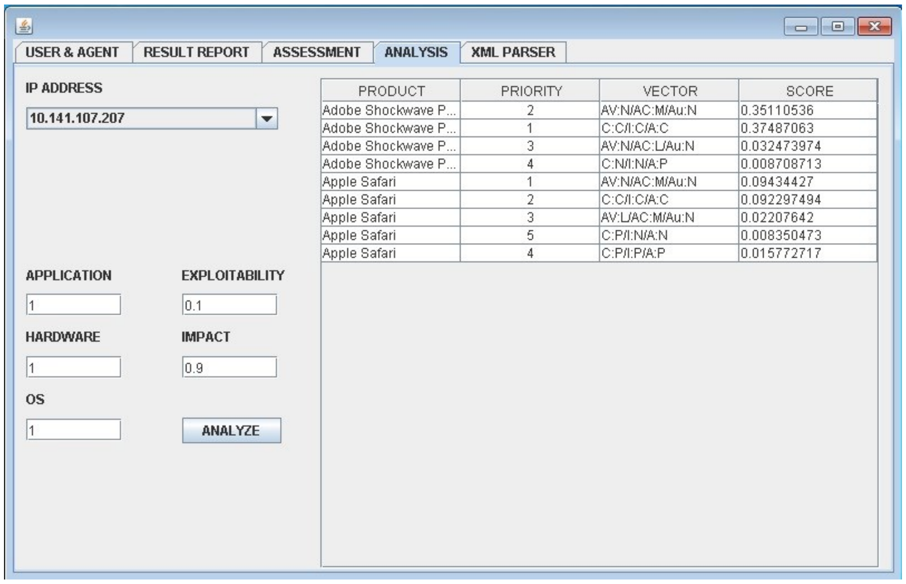
User Name: 10.141.107.207

Date: remote-8_27_5_12.xml

| Title | Content | Title |
|---|---|---|
| vulnerability | oval:org.mitre.oval:d... | WebKit Element Ru... |
| inventory | oval:org.mitre.oval:d... | Apple Safari is insta... |
| vulnerability | oval:org.mitre.oval:d... | Apple Safari Search ... |
| vulnerability | oval:org.mitre.oval:d... | Webkit Floating Poi... |
| vulnerability | oval:org.mitre.oval:d... | Apple Safari Local H... |
| vulnerability | oval:org.mitre.oval:d... | WebKit in Apple Saf... |
| vulnerability | oval:org.mitre.oval:d... | OPTIONS Request i... |

1) Username for IP address of the target system
2) Date and time for each document as XML file format
3) Detailed information of document for each file
4) Graphical analysis for selected document

- o Analysis Panel



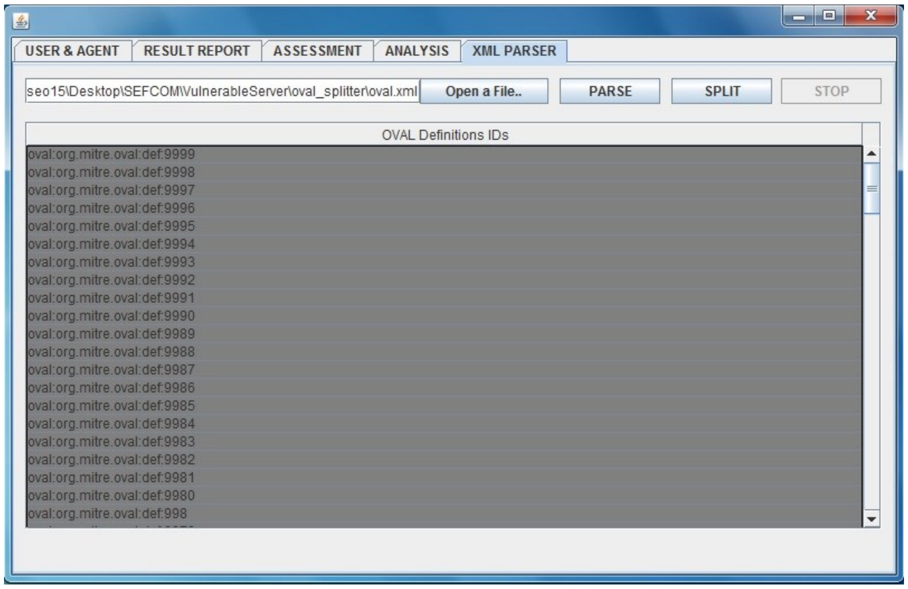
IP ADDRESS: 10.141.107.207

APPLICATION: 1
EXPLOITABILITY: 0.1
HARDWARE: 1
IMPACT: 0.9
OS: 1

| PRODUCT | PRIORITY | VECTOR | SCORE |
|---|---|---|---|
| Adobe Shockwave P... | 2 | AV:N/AC:M/Au:N | 0.35110536 |
| Adobe Shockwave P... | 1 | C:C/I:C/A:C | 0.37487063 |
| Adobe Shockwave P... | 3 | AV:N/AC:L/Au:N | 0.032473974 |
| Adobe Shockwave P... | 4 | C:N/I:N/A:P | 0.008708713 |
| Apple Safari | 1 | AV:N/AC:M/Au:N | 0.09434427 |
| Apple Safari | 2 | C:C/I:C/A:C | 0.092297494 |
| Apple Safari | 3 | AV:L/AC:M/Au:N | 0.02207642 |
| Apple Safari | 5 | C:P/I:N/A:N | 0.008350473 |
| Apple Safari | 4 | C:P/I:P/A:P | 0.015772717 |

1) Target system selection for analysis
2) Given weights from security administrator
3) Analysis result based on gathered information

- o XML Parser Panel



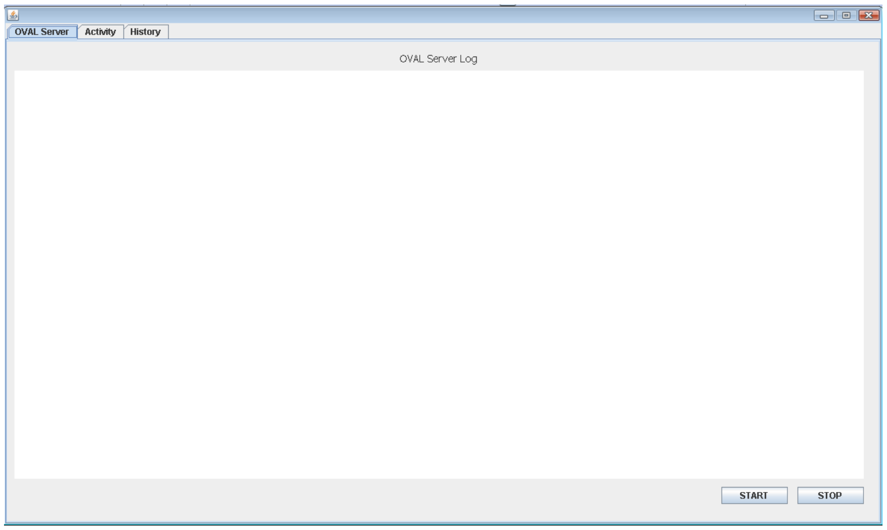
seo15\Desktop\SEFCOM\VulnerableServer\oval_splitter\oval.xml

OVAL Definitions IDs

```
oval:org.mitre.oval:def:9999
oval:org.mitre.oval:def:9998
oval:org.mitre.oval:def:9997
oval:org.mitre.oval:def:9996
oval:org.mitre.oval:def:9995
oval:org.mitre.oval:def:9994
oval:org.mitre.oval:def:9993
oval:org.mitre.oval:def:9992
oval:org.mitre.oval:def:9991
oval:org.mitre.oval:def:9990
oval:org.mitre.oval:def:9989
oval:org.mitre.oval:def:9988
oval:org.mitre.oval:def:9987
oval:org.mitre.oval:def:9986
oval:org.mitre.oval:def:9985
oval:org.mitre.oval:def:9984
oval:org.mitre.oval:def:9983
oval:org.mitre.oval:def:9982
oval:org.mitre.oval:def:9981
oval:org.mitre.oval:def:9980
oval:org.mitre.oval:def:998
```

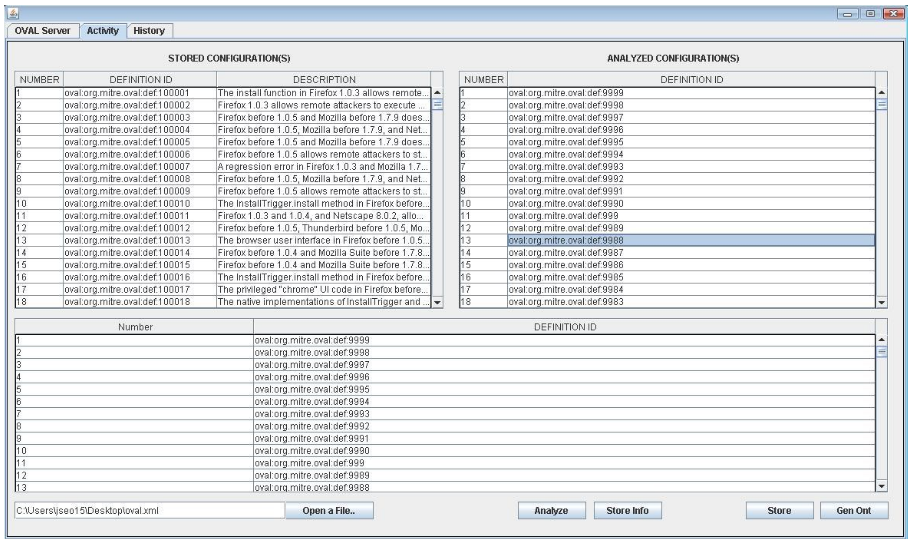
1) Pre-processing OVAL XML file to parse XML file into each definition

- OVAL Ontology Server
    - Log Panel



1) Log console for describing connections with target computers when server is started

- Activity Panel



STORED CONFIGURATION(S)

| NUMBER | DEFINITION ID | DESCRIPTION |
|---|---|---|
| 1 | oval:org.mitre.oval:def:100001 | The install function in Firefox 1.0.3 allows remote... |
| 2 | oval:org.mitre.oval:def:100002 | Firefox 1.0.3 allows remote attackers to execute ... |
| 3 | oval:org.mitre.oval:def:100003 | Firefox before 1.0.5 and Mozilla before 1.7.9 does... |
| 4 | oval:org.mitre.oval:def:100004 | Firefox before 1.0.5, Mozilla before 1.7.9, and Net... |
| 5 | oval:org.mitre.oval:def:100005 | Firefox before 1.0.5 and Mozilla before 1.7.9 does... |
| 6 | oval:org.mitre.oval:def:100006 | Firefox before 1.0.5 allows remote attackers to st... |
| 7 | oval:org.mitre.oval:def:100007 | A regression error in Firefox 1.0.3 and Mozilla 1.7... |
| 8 | oval:org.mitre.oval:def:100008 | Firefox before 1.0.5, Mozilla before 1.7.9, and Net... |
| 9 | oval:org.mitre.oval:def:100009 | Firefox before 1.0.5 allows remote attackers to st... |
| 10 | oval:org.mitre.oval:def:100010 | The InstallTrigger.install method in Firefox before... |
| 11 | oval:org.mitre.oval:def:100011 | Firefox 1.0.3 and 1.0.4, and Netscape 8.0.2, allo... |
| 12 | oval:org.mitre.oval:def:100012 | Firefox before 1.0.5, Thunderbird before 1.0.5, Mo... |
| 13 | oval:org.mitre.oval:def:100013 | The browser user interface in Firefox before 1.0.5... |
| 14 | oval:org.mitre.oval:def:100014 | Firefox before 1.0.4 and Mozilla Suite before 1.7.8... |
| 15 | oval:org.mitre.oval:def:100015 | Firefox before 1.0.4 and Mozilla Suite before 1.7.8... |
| 16 | oval:org.mitre.oval:def:100016 | The InstallTrigger.install method in Firefox before... |
| 17 | oval:org.mitre.oval:def:100017 | The privileged "chrome" UI code in Firefox before... |
| 18 | oval:org.mitre.oval:def:100018 | The native implementations of InstallTrigger and ... |

ANALYZED CONFIGURATION(S)

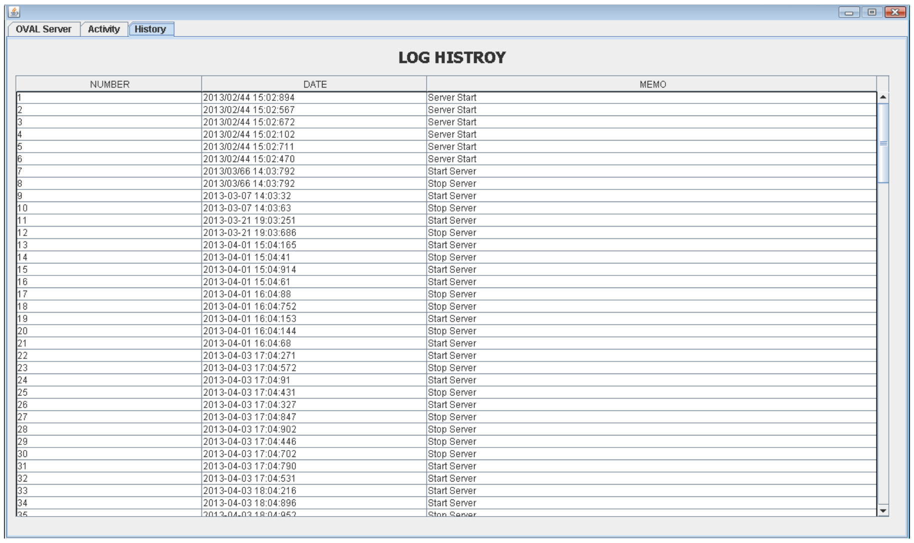
| NUMBER | DEFINITION ID |
|---|---|
| 1 | oval:org.mitre.oval:def:9999 |
| 2 | oval:org.mitre.oval:def:9998 |
| 3 | oval:org.mitre.oval:def:9997 |
| 4 | oval:org.mitre.oval:def:9996 |
| 5 | oval:org.mitre.oval:def:9995 |
| 6 | oval:org.mitre.oval:def:9994 |
| 7 | oval:org.mitre.oval:def:9993 |
| 8 | oval:org.mitre.oval:def:9992 |
| 9 | oval:org.mitre.oval:def:9991 |
| 10 | oval:org.mitre.oval:def:9990 |
| 11 | oval:org.mitre.oval:def:999 |
| 12 | oval:org.mitre.oval:def:9989 |
| 13 | oval:org.mitre.oval:def:9988 |
| 14 | oval:org.mitre.oval:def:9987 |
| 15 | oval:org.mitre.oval:def:9986 |
| 16 | oval:org.mitre.oval:def:9985 |
| 17 | oval:org.mitre.oval:def:9984 |
| 18 | oval:org.mitre.oval:def:9983 |

| Number | DEFINITION ID |
|---|---|
| 1 | oval:org.mitre.oval:def:9999 |
| 2 | oval:org.mitre.oval:def:9998 |
| 3 | oval:org.mitre.oval:def:9997 |
| 4 | oval:org.mitre.oval:def:9996 |
| 5 | oval:org.mitre.oval:def:9995 |
| 6 | oval:org.mitre.oval:def:9994 |
| 7 | oval:org.mitre.oval:def:9993 |
| 8 | oval:org.mitre.oval:def:9992 |
| 9 | oval:org.mitre.oval:def:9991 |
| 10 | oval:org.mitre.oval:def:9990 |
| 11 | oval:org.mitre.oval:def:999 |
| 12 | oval:org.mitre.oval:def:9989 |
| 13 | oval:org.mitre.oval:def:9988 |

C:\Users\jseo15\Desktop\oval.xml

1) Stored security standards displayed in Stored Configuration table
2) Analyzed configuration table shows newly analyzed OVAL XML file definition
3) Newly added definitions displayed in the bottom table

- History Panel



LOG HISTROY

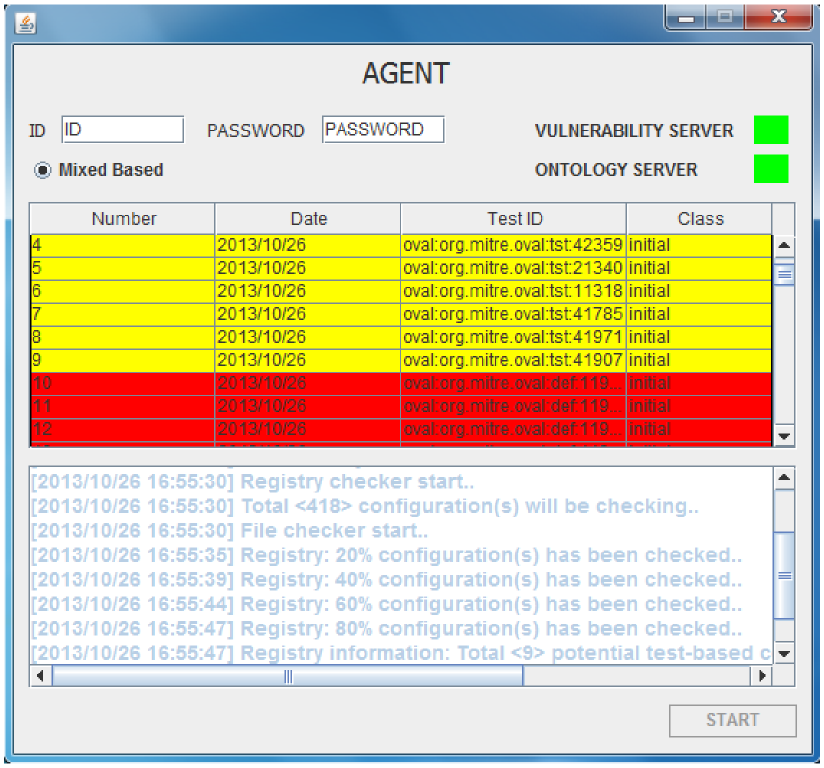
| NUMBER | DATE | MEMO |
|---|---|---|
| 1 | 2013/02/44 15:02:894 | Server Start |
| 2 | 2013/02/44 15:02:567 | Server Start |
| 3 | 2013/02/44 15:02:672 | Server Start |
| 4 | 2013/02/44 15:02:102 | Server Start |
| 5 | 2013/02/44 15:02:711 | Server Start |
| 6 | 2013/02/44 15:02:470 | Server Start |
| 7 | 2013/03/66 14:03:792 | Start Server |
| 8 | 2013/03/66 14:03:792 | Stop Server |
| 9 | 2013-03-07 14:03:32 | Start Server |
| 10 | 2013-03-07 14:03:63 | Stop Server |
| 11 | 2013-03-21 19:03:251 | Start Server |
| 12 | 2013-03-21 19:03:686 | Stop Server |
| 13 | 2013-04-01 15:04:165 | Start Server |
| 14 | 2013-04-01 15:04:41 | Stop Server |
| 15 | 2013-04-01 15:04:914 | Start Server |
| 16 | 2013-04-01 15:04:61 | Start Server |
| 17 | 2013-04-01 16:04:88 | Stop Server |
| 18 | 2013-04-01 16:04:752 | Stop Server |
| 19 | 2013-04-01 16:04:153 | Start Server |
| 20 | 2013-04-01 16:04:144 | Stop Server |
| 21 | 2013-04-01 16:04:68 | Start Server |
| 22 | 2013-04-03 17:04:271 | Start Server |
| 23 | 2013-04-03 17:04:572 | Stop Server |
| 24 | 2013-04-03 17:04:91 | Start Server |
| 25 | 2013-04-03 17:04:431 | Stop Server |
| 26 | 2013-04-03 17:04:327 | Start Server |
| 27 | 2013-04-03 17:04:847 | Stop Server |
| 28 | 2013-04-03 17:04:902 | Stop Server |
| 29 | 2013-04-03 17:04:446 | Stop Server |
| 30 | 2013-04-03 17:04:702 | Stop Server |
| 31 | 2013-04-03 17:04:790 | Start Server |
| 32 | 2013-04-03 17:04:531 | Start Server |
| 33 | 2013-04-03 18:04:216 | Start Server |
| 34 | 2013-04-03 18:04:896 | Start Server |
| 35 | 2013-04-03 18:04:952 | Stop Server |

1) Server start / stop history displayed

APPENDIX:

- AGENT

The agent monitors and captures suspicious events from the target system based on the given ID and Password. The Table shows detected vulnerability of test ids as yellow and definition ids as red. And the Console displays the processing procedure of agent tool.



AGENT

ID: ID  PASSWORD: PASSWORD  VULNERABILITY SERVER 🟩

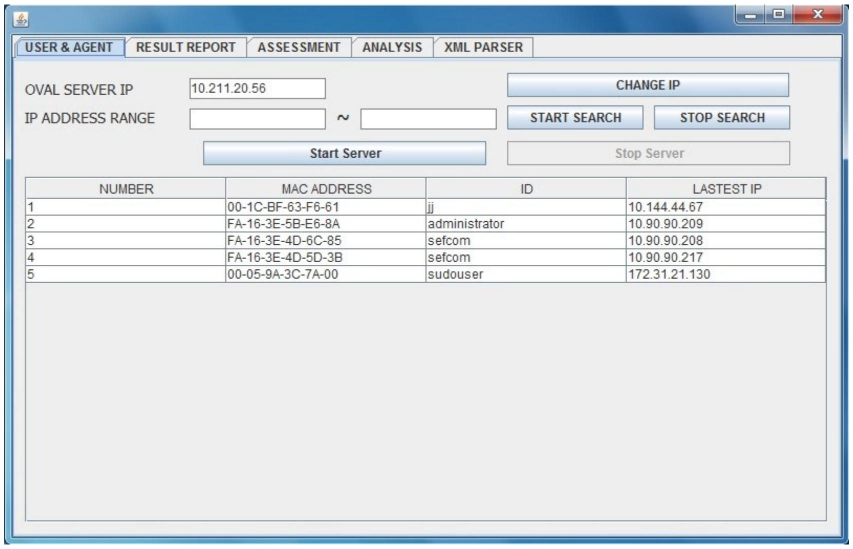
● Mixed Based  ONTOLOGY SERVER 🟩

| Number | Date | Test ID | Class |
|---|---|---|---|
| 4 | 2013/10/26 | oval:org.mitre.oval:tst:42359 | initial |
| 5 | 2013/10/26 | oval:org.mitre.oval:tst:21340 | initial |
| 6 | 2013/10/26 | oval:org.mitre.oval:tst:11318 | initial |
| 7 | 2013/10/26 | oval:org.mitre.oval:tst:41785 | initial |
| 8 | 2013/10/26 | oval:org.mitre.oval:tst:41971 | initial |
| 9 | 2013/10/26 | oval:org.mitre.oval:tst:41907 | initial |
| 10 | 2013/10/26 | oval:org.mitre.oval:def:119... | initial |
| 11 | 2013/10/26 | oval:org.mitre.oval:def:119... | initial |
| 12 | 2013/10/26 | oval:org.mitre.oval:def:119... | initial |

```
[2013/10/26 16:55:30] Registry checker start..
[2013/10/26 16:55:30] Total <418> configuration(s) will be checking..
[2013/10/26 16:55:30] File checker start..
[2013/10/26 16:55:35] Registry: 20% configuration(s) has been checked..
[2013/10/26 16:55:39] Registry: 40% configuration(s) has been checked..
[2013/10/26 16:55:44] Registry: 60% configuration(s) has been checked..
[2013/10/26 16:55:47] Registry: 80% configuration(s) has been checked..
[2013/10/26 16:55:47] Registry information: Total <9> potential test-based c
```

- Vulnerability Server
    - User and Agent Panel

      Through the User and Agent Panel, administrator can set up the OVAL Ontology server IP to provide an address to the agent installed in the target computer. Server can trace agent tool over the network by given IP range and display detected agent in IP range.



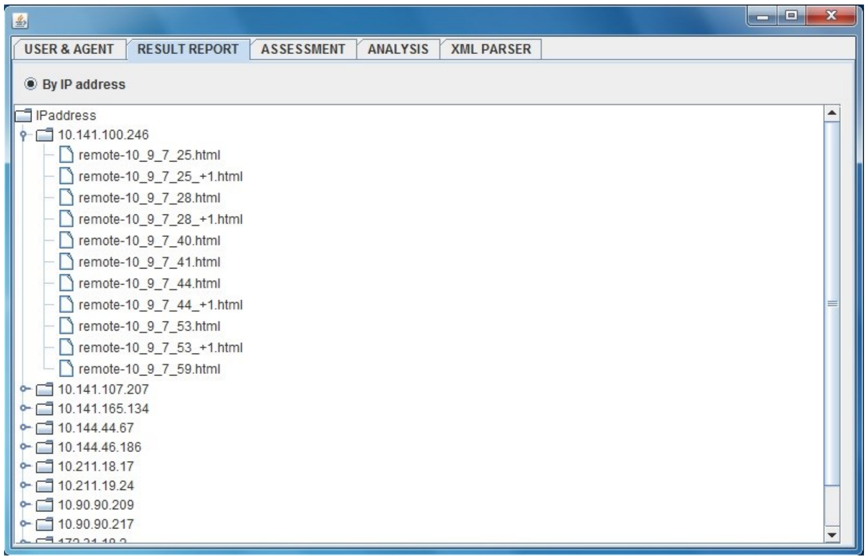
OVAL SERVER IP 10.211.20.56

IP ADDRESS RANGE ~

| NUMBER | MAC ADDRESS | ID | LASTEST IP |
|---|---|---|---|
| 1 | 00-1C-BF-63-F6-61 | jj | 10.144.44.67 |
| 2 | FA-16-3E-5B-E6-8A | administrator | 10.90.90.209 |
| 3 | FA-16-3E-4D-6C-85 | sefcom | 10.90.90.208 |
| 4 | FA-16-3E-4D-5D-3B | sefcom | 10.90.90.217 |
| 5 | 00-05-9A-3C-7A-00 | sudouser | 172.31.21.130 |

    - Result Report Panel

      Result report panel displays detected vulnerabilities in the target system with its detection time. When a security administrator double clicks each attribute, the reported result of detected vulnerability will be provided.

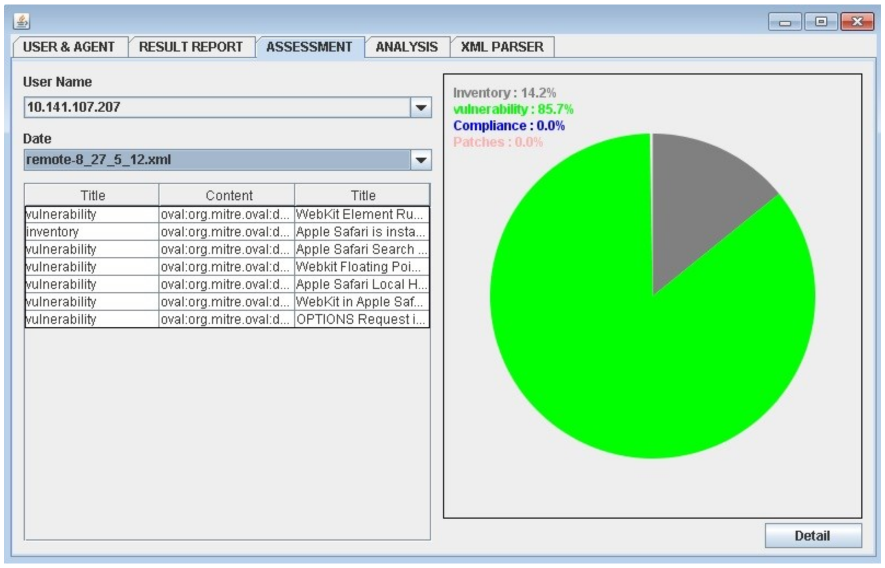


By IP address

```
IPaddress
  10.141.100.246
    remote-10_9_7_25.html
    remote-10_9_7_25_+1.html
    remote-10_9_7_28.html
    remote-10_9_7_28_+1.html
    remote-10_9_7_40.html
    remote-10_9_7_41.html
    remote-10_9_7_44.html
    remote-10_9_7_44_+1.html
    remote-10_9_7_53.html
    remote-10_9_7_53_+1.html
    remote-10_9_7_59.html
  10.141.107.207
  10.141.165.134
  10.144.44.67
  10.144.46.186
  10.211.18.17
  10.211.19.24
  10.90.90.209
  10.90.90.217
```

- Assessment Panel

  The assessment panel gives graphical analysis of each document containing results of the detected vulnerability in the target system. Table shows vulnerabilities occurred in the target system and graph reflects vulnerability result of each document.



User Name: 10.141.107.207

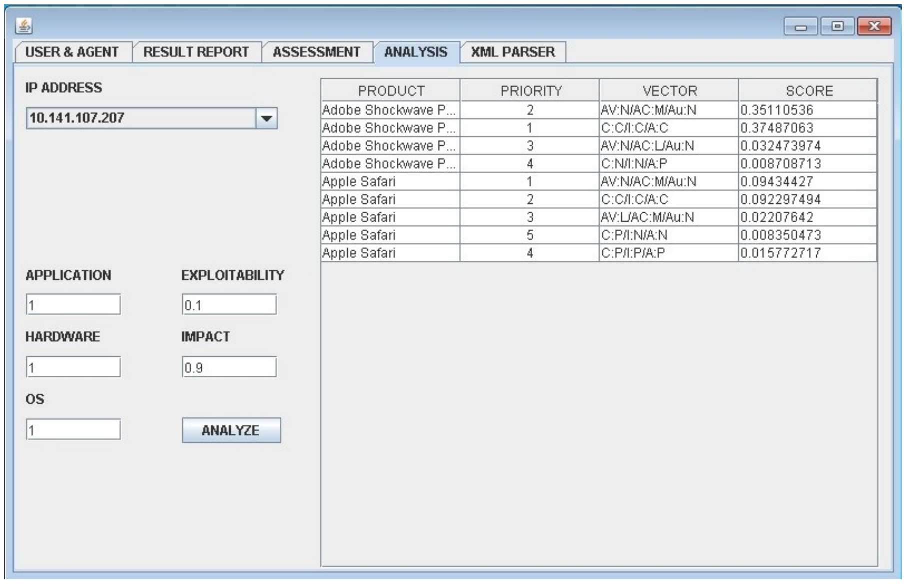
Date: remote-8_27_5_12.xml

| Title | Content | Title |
|---|---|---|
| vulnerability | oval:org.mitre.oval:d... | WebKit Element Ru... |
| inventory | oval:org.mitre.oval:d... | Apple Safari is insta... |
| vulnerability | oval:org.mitre.oval:d... | Apple Safari Search ... |
| vulnerability | oval:org.mitre.oval:d... | Webkit Floating Poi... |
| vulnerability | oval:org.mitre.oval:d... | Apple Safari Local H... |
| vulnerability | oval:org.mitre.oval:d... | WebKit in Apple Saf... |
| vulnerability | oval:org.mitre.oval:d... | OPTIONS Request i... |

Inventory : 14.2%
vulnerability : 85.7%
Compliance : 0.0%
Patches : 0.0%

- Analysis Panel

  Based on given IP address, the server collects information which are the detected vulnerabilities in the target system. It calculates priority of the vulnerabilities by the given weights from security administrator at the table.



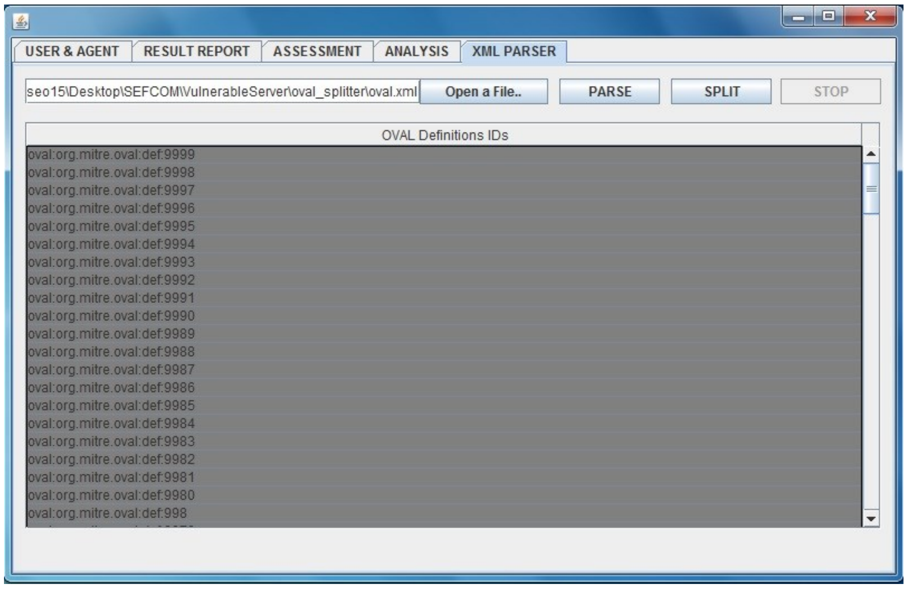
IP ADDRESS: 10.141.107.207

APPLICATION: 1
EXPLOITABILITY: 0.1
HARDWARE: 1
IMPACT: 0.9
OS: 1

| PRODUCT | PRIORITY | VECTOR | SCORE |
|---|---|---|---|
| Adobe Shockwave P... | 2 | AV:N/AC:M/Au:N | 0.35110536 |
| Adobe Shockwave P... | 1 | C:C/I:C/A:C | 0.37487063 |
| Adobe Shockwave P... | 3 | AV:N/AC:L/Au:N | 0.032473974 |
| Adobe Shockwave P... | 4 | C:N/I:N/A:P | 0.008708713 |
| Apple Safari | 1 | AV:N/AC:M/Au:N | 0.09434427 |
| Apple Safari | 2 | C:C/I:C/A:C | 0.092297494 |
| Apple Safari | 3 | AV:L/AC:M/Au:N | 0.02207642 |
| Apple Safari | 5 | C:P/I:N/A:N | 0.008350473 |
| Apple Safari | 4 | C:P/I:P/A:P | 0.015772717 |

- XML Parser Panel

  The server parses and splits each vulnerability definition based on OVAL XML file in given path. This pre-processing helps performance of vulnerability server to verify existing vulnerability in the target system. If the definition already exists in the specific path, it will show id column with gray color background.



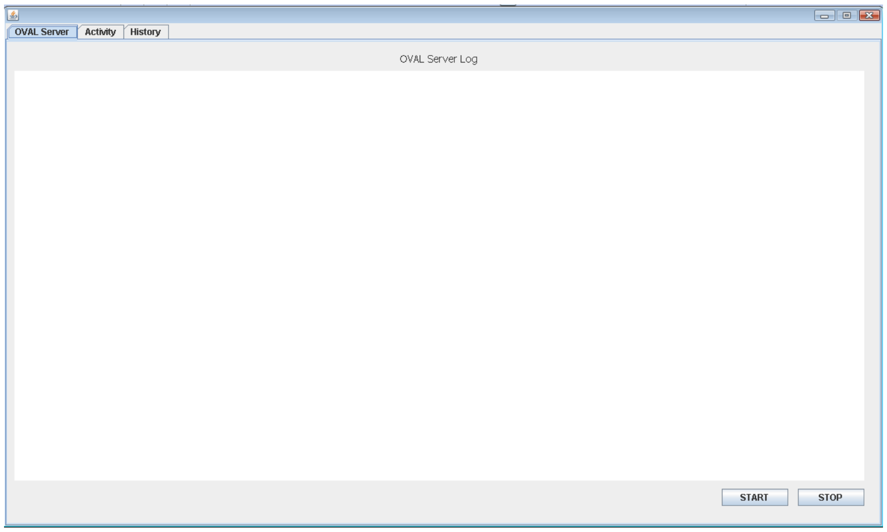
```
seo15\Desktop\SEFCOM\VulnerableServer\oval_splitter\oval.xml
```

OVAL Definitions IDs

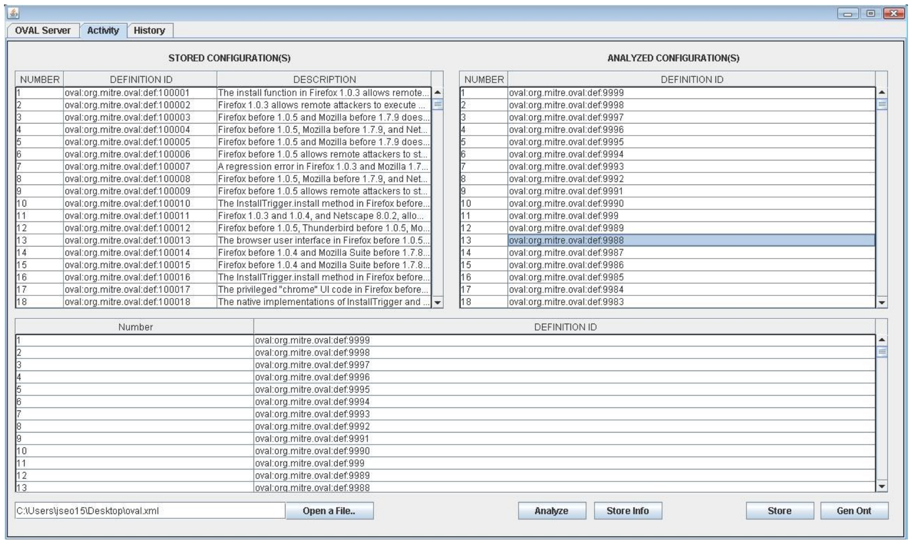
```
oval:org.mitre.oval:def:9999
oval:org.mitre.oval:def:9998
oval:org.mitre.oval:def:9997
oval:org.mitre.oval:def:9996
oval:org.mitre.oval:def:9995
oval:org.mitre.oval:def:9994
oval:org.mitre.oval:def:9993
oval:org.mitre.oval:def:9992
oval:org.mitre.oval:def:9991
oval:org.mitre.oval:def:9990
oval:org.mitre.oval:def:9989
oval:org.mitre.oval:def:9988
oval:org.mitre.oval:def:9987
oval:org.mitre.oval:def:9986
oval:org.mitre.oval:def:9985
oval:org.mitre.oval:def:9984
oval:org.mitre.oval:def:9983
oval:org.mitre.oval:def:9982
oval:org.mitre.oval:def:9981
oval:org.mitre.oval:def:9980
oval:org.mitre.oval:def:998
```

- OVAL Ontology Server
  - Log Panel

    Logs reveal when and which agent tool requested the vulnerability information in the network.



- Activity Panel
  The parsed definitions from OVAL XML file are stored in the database. 'stored configurations' tab shows stored information. Newly analyzed information based on give path of OVAL XML file is displayed in analyzed configuration and definitions which are not stored in the database are visible in the bottom table.



STORED CONFIGURATION(S)

| NUMBER | DEFINITION ID | DESCRIPTION |
|---|---|---|
| 1 | oval:org.mitre.oval:def:100001 | The install function in Firefox 1.0.3 allows remote... |
| 2 | oval:org.mitre.oval:def:100002 | Firefox 1.0.3 allows remote attackers to execute ... |
| 3 | oval:org.mitre.oval:def:100003 | Firefox before 1.0.5 and Mozilla before 1.7.9 does... |
| 4 | oval:org.mitre.oval:def:100004 | Firefox before 1.0.5, Mozilla before 1.7.9, and Net... |
| 5 | oval:org.mitre.oval:def:100005 | Firefox before 1.0.5 and Mozilla before 1.7.9 does... |
| 6 | oval:org.mitre.oval:def:100006 | Firefox before 1.0.5 allows remote attackers to st... |
| 7 | oval:org.mitre.oval:def:100007 | A regression error in Firefox 1.0.3 and Mozilla 1.7... |
| 8 | oval:org.mitre.oval:def:100008 | Firefox before 1.0.5, Mozilla before 1.7.9, and Net... |
| 9 | oval:org.mitre.oval:def:100009 | Firefox before 1.0.5 allows remote attackers to st... |
| 10 | oval:org.mitre.oval:def:100010 | The InstallTrigger.install method in Firefox before... |
| 11 | oval:org.mitre.oval:def:100011 | Firefox 1.0.3 and 1.0.4, and Netscape 8.0.2, allo... |
| 12 | oval:org.mitre.oval:def:100012 | Firefox before 1.0.5, Thunderbird before 1.0.5, Mo... |
| 13 | oval:org.mitre.oval:def:100013 | The browser user interface in Firefox before 1.0.5... |
| 14 | oval:org.mitre.oval:def:100014 | Firefox before 1.0.4 and Mozilla Suite before 1.7.8... |
| 15 | oval:org.mitre.oval:def:100015 | Firefox before 1.0.4 and Mozilla Suite before 1.7.8... |
| 16 | oval:org.mitre.oval:def:100016 | The InstallTrigger.install method in Firefox before... |
| 17 | oval:org.mitre.oval:def:100017 | The privileged "chrome" UI code in Firefox before... |
| 18 | oval:org.mitre.oval:def:100018 | The native implementations of InstallTrigger and ... |

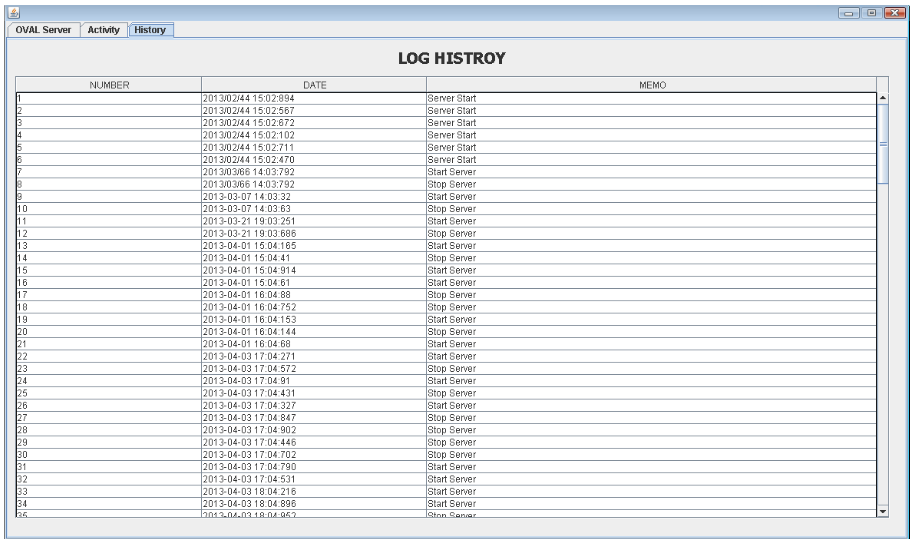
ANALYZED CONFIGURATION(S)

| NUMBER | DEFINITION ID |
|---|---|
| 1 | oval:org.mitre.oval:def:9999 |
| 2 | oval:org.mitre.oval:def:9998 |
| 3 | oval:org.mitre.oval:def:9997 |
| 4 | oval:org.mitre.oval:def:9996 |
| 5 | oval:org.mitre.oval:def:9995 |
| 6 | oval:org.mitre.oval:def:9994 |
| 7 | oval:org.mitre.oval:def:9993 |
| 8 | oval:org.mitre.oval:def:9992 |
| 9 | oval:org.mitre.oval:def:9991 |
| 10 | oval:org.mitre.oval:def:9990 |
| 11 | oval:org.mitre.oval:def:999 |
| 12 | oval:org.mitre.oval:def:9989 |
| 13 | oval:org.mitre.oval:def:9988 |
| 14 | oval:org.mitre.oval:def:9987 |
| 15 | oval:org.mitre.oval:def:9986 |
| 16 | oval:org.mitre.oval:def:9985 |
| 17 | oval:org.mitre.oval:def:9984 |
| 18 | oval:org.mitre.oval:def:9983 |

| Number | DEFINITION ID |
|---|---|
| 1 | oval:org.mitre.oval:def:9999 |
| 2 | oval:org.mitre.oval:def:9998 |
| 3 | oval:org.mitre.oval:def:9997 |
| 4 | oval:org.mitre.oval:def:9996 |
| 5 | oval:org.mitre.oval:def:9995 |
| 6 | oval:org.mitre.oval:def:9994 |
| 7 | oval:org.mitre.oval:def:9993 |
| 8 | oval:org.mitre.oval:def:9992 |
| 9 | oval:org.mitre.oval:def:9991 |
| 10 | oval:org.mitre.oval:def:9990 |
| 11 | oval:org.mitre.oval:def:999 |
| 12 | oval:org.mitre.oval:def:9989 |
| 13 | oval:org.mitre.oval:def:9988 |

C:\Users\jseo15\Desktop\oval.xml

- History Panel
  Log History table logs the server start / stop by security administrator.



LOG HISTROY

| NUMBER | DATE | MEMO |
|---|---|---|
| 1 | 2013/02/44 15:02:894 | Server Start |
| 2 | 2013/02/44 15:02:567 | Server Start |
| 3 | 2013/02/44 15:02:672 | Server Start |
| 4 | 2013/02/44 15:02:102 | Server Start |
| 5 | 2013/02/44 15:02:711 | Server Start |
| 6 | 2013/02/44 15:02:470 | Server Start |
| 7 | 2013/03/66 14:03:792 | Start Server |
| 8 | 2013/03/66 14:03:792 | Stop Server |
| 9 | 2013-03-07 14:03:32 | Start Server |
| 10 | 2013-03-07 14:03:63 | Stop Server |
| 11 | 2013-03-21 19:03:251 | Start Server |
| 12 | 2013-03-21 19:03:686 | Stop Server |
| 13 | 2013-04-01 15:04:165 | Start Server |
| 14 | 2013-04-01 15:04:41 | Stop Server |
| 15 | 2013-04-01 15:04:914 | Start Server |
| 16 | 2013-04-01 15:04:61 | Start Server |
| 17 | 2013-04-01 16:04:88 | Stop Server |
| 18 | 2013-04-01 16:04:752 | Stop Server |
| 19 | 2013-04-01 16:04:153 | Start Server |
| 20 | 2013-04-01 16:04:144 | Stop Server |
| 21 | 2013-04-01 16:04:68 | Start Server |
| 22 | 2013-04-03 17:04:271 | Start Server |
| 23 | 2013-04-03 17:04:572 | Stop Server |
| 24 | 2013-04-03 17:04:91 | Start Server |
| 25 | 2013-04-03 17:04:431 | Stop Server |
| 26 | 2013-04-03 17:04:327 | Start Server |
| 27 | 2013-04-03 17:04:847 | Stop Server |
| 28 | 2013-04-03 17:04:902 | Stop Server |
| 29 | 2013-04-03 17:04:446 | Stop Server |
| 30 | 2013-04-03 17:04:702 | Stop Server |
| 31 | 2013-04-03 17:04:790 | Start Server |
| 32 | 2013-04-03 17:04:531 | Start Server |
| 33 | 2013-04-03 18:04:216 | Start Server |
| 34 | 2013-04-03 18:04:896 | Start Server |
| 35 | 2013-04-03 18:04:952 | Stop Server |