APPROVAL SHEET

Title of Thesis: SCAP compliant android vulnerability scanner

Name of Candidate: Rujuta S. Palande
Masters of Science, 2017

Thesis and Abstract Approved: _____

Dr. Charles Nicholas
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

ABSTRACT

Title of Thesis: SCAP compliant android vulnerability scanner

Rujuta Palande, Masters of Science, 2017

Thesis directed by: Dr. Charles Nicholas, Professor
Department of Computer Science and
Electrical Engineering

This thesis attempts to explain the SCAP compliance of a preliminary vulnerability scanner which is in the form of an OVAL interpreter (and thus, SCAP compliant) , which scans for vulnerabilities reported because of the telephony feature in android, in the year 2016 as reported in the national vulnerability database (NVD). The implementation of the scanner is achieved by attempting to write an OVAL definition file, which when evaluated against a system characteristics file , produces an OVAL results file. The result file thus generated is in a standard form , which can be understood and interpreted by other SCAP compliant scanners as well, thus ensuring interoperability and standardization.

SCAP compliant android vulnerability scanner

by

Rujuta S. Palande

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Masters of Science
2017

I dedicate my work to aai, baba, aaji, aaba, mama, mami, pranav and tejas

ACKNOWLEDGMENTS

I would like to first express my deepest gratitude to my thesis advisor, Dr. Charles Nicholas for supporting me through my master's study and research. I am gratefully indebted to his invaluable guidance, understanding, patience and motivation for this thesis. I would also like to thank my parents Shrinivas Palande and Yogini Palande, my brothers,Pranav and Tejas, my uncle and aunt- Nishikant Joshi and Dhanashree Joshi and my grandparents Pratibha Palande, Rohini Joshi, Vinayak Joshi and late Mr. Anant Palande for being my strength. This accomplishment would not have been possible without them. Thank you!

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 What is a vulnerability?

According to IETF RFC 2828 as mentioned in (Xi-Salvador 2008) a vulnerability is a flaw or a weakness which can be found in a particular system's design, implementation, or operation and management that could be exploited by a threat, to violate the system's security policy. Some common types of software flaws that lead to vulnerabilities are buffer overflows, dangling pointers, SQL injections, race conditions, etc.

If a computer system is imagined to be an amalgamation of states describing the current configuration of that system, all states reachable from a given initial state using a set of authorized state transitions fall into the class of either authorized or unauthorized states as defined by a security policy for that system. A vulnerable state is an authorized state from which an unauthorized state can be reached by using a series of authorized state transitions. This results in the system reaching to a compromised state or a vulnerable state. An attack is a sequence of authorized state transitions which end in this compromised state. We come across many other terms in Computer Security like malware, trojan, virus, etc. with which, a software vulnerability is often confused. But all the above terms have a different meaning and significance in the domain of computer security.

- A Malware is a software which is specifically written with the intent to harm a system.

- A virus is a specific type of malware that spreads itself once it is initially run.
- A worm is a virus that is self-contained. It does not attach itself like a parasite. Instead, it goes around searching out other machines to infect.
- A Trojan is a type of malware that is often disguised as legitimate software. Trojans can be employed by cyber-thieves and hackers trying to gain access to users' systems.

1.1.1 Consequences of vulnerabilities

Computing systems today have substantial number of security configuration settings that are designed to offer flexible and robust services. However, incorrect configuration increases the potential of vulnerability and attacks. The range of threats that can result because of the exploitation of these vulnerabilities is enormous, and may result into possible terrorist activities, sabotage,copycat crimes, mechanical malfunctions, and human error. Attacks may involve Trojan-horse insertion and physical tampering, including destructive acts by unhappy and dissatisfied employees or former employees. Denial of service attacks are particularly dangerous, because they are so difficult to defend against and because their effects can be devastating. Systems connected to the Internet or available by dial-up lines are potential victims of external penetrations. Even systems that appear to be completely isolated are subjected to internal misuse. So basically, vulnerabilities can have devastating consequences and they need to be detected and acted upon.

1.1.2 The concept of vulnerability scanners

Vulnerability scanning is the process of inspection of the potential points of exploit on a computer or network to identify security holes which are an entry point to any possible threats. A vulnerability scan detects and classifies system weaknesses or the entry points for threats in computers, networks and communications equipment and predicts the effectiveness of their remedies.

1.2 Need for standardization

Misconfigurations and vulnerabilities can be identified and globally categorized only if global checking that includes network and desktop configuration is performed, as many of these configurations are dependent on each other. This induces the need of a collective understanding of WHAT as correctly stated by (Schmidt)

- Are we talking about the same software vulnerability or a misconfiguration?

- Do we agree on what a policy recommendation means and how to meet it?

These are really tough questions without the existence of standards. Content authors strive to provide a solution to identify vulnerabilities and misconfigurations on a particular system. But if the content is not universal, they might end up writing for each assessment tool. With the standardization, content authors dont need to write for each assessment tool . They can establish a shared content repository everyone can use,with a consistent understanding.

1.2.1 SCAP to the rescue

The Security Content Automation Protocol (SCAP), provides a standard approach to maintain the security of enterprise systems by automatically verifying the presence of patches, checking system security configuration settings, examining systems for any vulnerabilities, etc. SCAP was defined and started by the National Institute of Standards and Technology (NIST) and its partners in industry. It is basically a super-standard consisting of many individually maintained standards.The SCAP suite of specifications standardize the nomenclature and formats used by the automated vulnerability management and policy compliance products. Applications which conduct security monitoring, use these standards when considering systems to find vulnerabilities, and offer methods to score those findings according to the severity of the vulnerability in order to evaluate the possible impact. A vendor of a computer system configuration scanner can get his product validated against

SCAP, demonstrating that it will interoperate with other scanners, and express the scan results in a standardized way.

The latest version of SCAP: SCAP 1.3 has around 12 different specifications as follows:

1. Languages

    - XCCDF: According to (Worrell) XCCDF (The Extensible Configuration Checklist Description Format) is a test definition language for writing security checklists, benchmarks, etc. An XCCDF document represents a structured collection of security configuration rules for some set of target systems.
    - OVAL: The Open Vulnerability and Assessment Language (OVAL) standardizes the three main steps of the assessment process: representing configuration information of systems for testing, analyzing the system for the presence of the specified machine state (vulnerability, configuration, patch state, etc.) and reporting the results of this assessment.
    - OCIL: The Open Checklist Interactive Language (OCIL) defines a framework for expressing a set of questions to be presented to a user and corresponding procedures to interpret responses to these questions.
    - Asset Identification: The Asset Identifier is used to uniquely identify assets based on known identifiers or known information about the assets. It plays an important role in an organization's ability to quickly correlate different sets of information about assets.
    - ARF: The Asset Reporting Format (ARF) is a data model to express the transport format of information about assets, and the relationships between assets and reports. The standardized data model facilitates the reporting, correlating, and fusing of asset information throughout and between organizations.

2. Identification schemes:

    - CCE: Common Configuration Enumeration (CCE) provides unique identifiers for configuration issues to facilitate correlation of configuration data across multiple information sources and tools.
    - CPE: Common Platform Enumeration (CPE) is a standardized method of describing and identifying classes of applications, operating systems, and hardware devices present among an enterprise's computing assets. It is used for mapping platforms to vulnerabilities or policy statements.
    - CVE: Common Vulnerabilities and Exposures (CVE) is a dictionary of common names or identifiers for publicly known information security vulnerabilities. If a CVE identifier is found in any of the reports of security tools, then the remedy for it can be searched in the CVE compatible databases.
    - SWID: Software identification (SWID) tags define a structured metadata format that identify the software product, characterize the product's version, the organizations and individuals that had a role in the production and distribution of the product etc.

3. Metrics

    - CVSS: The Common Vulnerability Scoring System (CVSS) algorithm scores a given vulnerability based on its likely danger on a scale of 0 to 10. It is mainly used for prioritizing responses to the detected vulnerabilities and weighing the cost of remedies for that vulnerability against allowing a vulnerability to persist.
    - CCSS: Similar to CVSS, The Common Configuration Scoring System (CCSS) is a set of measures of the severity of software security configuration issues.

4. Integrity

- TMSAD: The Trust Model for Security Automation Data (TMSAD) describes a common trust model which is composed of recommendations on how to use existing specifications to represent signatures, hashes, key information, and identity information in the context of an XML document within the security automation domain.

1.2.2 SCAP Validation program

The SCAP Validation Program tests whether the products comply to SCAP standards. The NIST National Voluntary Laboratory Accreditation Program (NVLAP) gives accredition to independent laboratories under the program to perform SCAP validations on different products by commercial vendors.A vendor seeking validation of his security product can contact an NVLAP accredited SCAP validation laboratory for assistance in the validation process or can get his product SCAP validated.

1.3 What is OVAL?

Determining the existence of software vulnerabilities, configuration issues, programs, and patches in local systems had no structured means because of which system administrators and other end users faced problems. They were restricted to analyzing text-based descriptions which was a labor-intensive and error-prone process to determine if any issue existed on the local system. OVAL came to the rescue to solve these problems. As mentioned previously, The Open Vulnerability and Assessment Language (OVAL) standardizes the three main steps of the assessment process: representing configuration information of systems for testing, analyzing the system for the presence of the specified machine state (vulnerability, configuration, patch state, etc.) and reporting the results of this assessment. The widespread availability of OVAL promotes standardized vulnerability and configura-

tion assessment and will provide consistent and reproducible information. OVAL helps you to determine which vulnerabilities or configuration issues exist on your system. You may then use this information to obtain appropriate software patches and fix information for remediation. Hence, OVAL acts as a preventive measure. As an addition, you can use a vulnerability database which includes information about vulnerabilities that OVAL does not, such as the severity of the problem, whether it is locally or remotely exploitable, remediation information, and so on. OVAL has been used to determine the vulnerabilities on various platforms like Windows, RHEL, Solaris, etc. The OVAL community has developed three schemas written in Extensible Markup Language (XML) to serve as the framework and vocabulary of the OVAL Language. These schemas correspond to the three steps of the assessment process: an OVAL System Characteristics schema for representing system information, an OVAL Definition schema for expressing a specific machine state, and an OVAL Results schema for reporting the results of an assessment. Content written in the OVAL Language is located in one of the many repositories found within the community. One such repository, known as the OVAL Repository, is hosted by The MITRE Corporation. It is the central meeting place for the OVAL Community to discuss, analyze, store, and disseminate OVAL Definitions. Each definition in the OVAL Repository determines whether a specified software vulnerability, configuration issue, program, or patch is present on a system.

1.3.1 The OVAL Interpreter

This is a basic reference implementation created to show how data can be collected from a computer for testing based on a set of OVAL Definitions and then evaluated to determine the results of each definition. It is not a fully functional scanning tool and has a simple user interface, but running the OVAL Interpreter provides with a list of result values for each evaluated definition, in a standard format which can be understood and processed

by other SCAP compliant scanners.

1.4 Vulnerabilities in android

Android is a truly open OS, and that makes it risky. Google Play (formerly called the Android Market), the digital distribution platform for applications for Android devices, is itself a source of potential security risks. When users download apps from Google Play, they often don't pay attention to the extent of permissions an app can have on their device. The security vulnerabilities affecting Android devices (Android vulnerabilities by Google) can cause actual performance issues and data loss and not just minor inconveniences. Inability of the Android operating system to cope with the vulnerabilities and malicious exploits has been spoken of and backed by research, many a times. According to a research conducted by the UK's University of Cambridge, in 2015 (Thomas, Beresford, & Rice 2015), nearly 90 percent of Android devices are exposed to at least one critical vulnerability. Some of the dangerous vulnerabilities include

- Dirty Cow vulnerability (Gurfinkel 2016) which exploits a mechanism called copy-on-write, allowing an attacker to gain privilege escalation on the Linux kernel.
- StageFright vulnerability (Also known as the mother of all android vulnerabilities) which allows an attacker to perform arbitrary operations on the victim's device through remote code execution and privilege escalation
- "Fake ID which allows malicious apps to associate themselves with certificates from legitimate apps, thus gaining access to stuff they shouldn't have access to.

Some of these vulnerabilities result into hazardous consequences. For eg., A malicious hacker could take advantage of it by sending to an Android device a straightforward text message that, once received by the smartphone, would give him complete control over the handset and allow him to steal anything on it, such as credit card numbers or personal

information.

1.5 The thesis idea

As can be concluded from the above discussion, android vulnerabilities are dangerous and can cause hazardous effects if exploited. So this made me explore the different android vulnerability scanners available in google play store and after some preliminary research, I concluded that none of the vulnerability scanners are SCAP compliant. I then went on to research every SCAP compliant product which was mentioned on the official website of SCAP and skimmed through the supported platforms for each product and found that none of them support the android OS. So I decided to work on writing a basic SCAP compliant preliminary android vulnerability scanner wherein I used OVAL to write definitions for a subset of vulnerabilities mentioned in the NVD (national vulnerability database) and construct a basic OVAL interpreter which will scan the android operating system for these specific vulnerabilities. The results are produced in the format mentioned on the mitre.org site which is an open community for OVAL. Thus this scanner aims at being an SCAP compliant scanner as it adheres to the OVAL standards and uses the standard CVE and CCE annotations. The thesis further describes the research carried out, the implementation of the scanner, the evaluation tests conducted and their comparison to the ground truth.

Chapter 2

RELATED WORK

Though not much work has been done in the area of developing an OVAL interpreter specifically for android (though schemas for android have been provided by mitre.org in December 2014) and attempting to make an android vulnerability scanner which is SCAP compliant, much work has been done in the domain of vulnerability reporting and scanning for android. Also, many vendors have developed their implementation of the OVAL interpreter but for operating systems other than android. This section discusses about the work done in the above-mentioned areas.

2.0.1 Vulnerability scanners in android

Android has been susceptible to many vulnerabilities. Some occur at the system level while some, at the application level.As mentioned earlier, just under nine in ten Android devices (87.7 percent) are exposed to at least one of the eleven critical vulnerabilities, which is an alarming number indeed. The study considered, concerned itself with vulnerabilities that gave an attacker significant permissions such as root level access without having physical access to the device. These could be through an installed application, dynamic code, or code injection. Attempts have been made to detect these vulnerabilities well in advance before any malicious software or attacker can use the knowledge of that vulnerability and

penetrate a mobile phone. Below are some of the vulnerability scanners and the different areas in which they predominantly operate :

1. Checkpoint quadrooter scanner

   Quadrooter is a group of four system level vulnerabilities affecting those android devices which have QUALCOMM chipset and associated driver code (point research team). The four vulnerabilities defined as quadrooter are :

   - CVE-2016-2503
   - CVE-2016-2504
   - CVE-2016-2509
   - CVE-2016-5340

   The details of these vulnerabilities are mentioned in the national vulnerability database (NVD). These vulnerabilities reside in the embedded software running the graphics driver. The graphics driver has privileged access to other processes on the android device which makes it possible for any ill-intended hacker to attack. So basically the attacker can gain privileges through a crafted application and take complete control of android devices, potentially exposing your sensitive data to cyber crime. Googles android security bulletin for July and August 2016 mentions about this vulnerability while also stating that until now, they do not have any reports of active customer exploitation of abuse of these reported issues.

   The checkpoint quadrooter scanner analyses the android smartphones or tablets to discover if its vulnerable to the quadrooter vulnerabilities. The scanner app is designed to give a clear indication of the threat risk to the android device and provides more information about QuadRooter, including which vulnerabilities affect the specific device and how they work.

2. Stagefright detector by Zimperium (Stagefright)

The stagefright vulnerability , also known as Worst android bug in the history and mother of all android vulnerabilities, has the potential of affecting a billion phones as this vulnerability exists in the operating system itself. It allows cyber-criminals to hack an android phone in less than 10 seconds. The newest version of Stagefright, also referred to as Metaphor, tricks a user into visiting a hacker's web page, containing a malicious multimedia file that when received, infects a phone. Once a malicious message is downloaded, it resets a phone and forces it to send a unique video file to the device. Using this data, a hacker can take control of a device to gain access to personal information, as well as being able to copy data and use the microphone and camera.

This exploit was announced by mobile security firm : Zipmerium, as a part of an announcement for its annual party at the BlackHat conference. Zimperium zLabs (Zlabs) created a Detector app to validate that you are running a version that is not vulnerable to the discovered Stagefright vulnerabilities. Some phone vendors have released partial patches to the vulnerabilities disclosed and this app can help you to understand if your device is vulnerable or not. This application tells you three major things :

- Whether your device is vulnerable
- Which CVEs your device is vulnerable to
- Whether you need to update your mobile operating system

The above two scanners are basically operating system level vulnerability scanners. A few scanners have also been developed which are application vulnerabilities scanners.

3. AppVigil

This application vulnerability scanner helps identification of security and privacy

vulnerabilities during application development lifecycle. It detects vulnerabilities at a production level when an application is being developed, and also gives patch recommendations. This comes as an add-on to android studio as well as eclipse development environment. Appvigil is a completely automated Mobile Reputation Protection Suite for Mobile Apps. Powered by patent pending technology, Appvigil employs intensive static analysis and dynamic analysis along with stringent network analysis on the apps. It discovers security vulnerabilities in mobile apps, delivering thorough and comprehensive reports in just few hours.The report points to the exact location of the security threat, along its descriptions and recommendations. It helps you make effective modifications and plugging loopholes in your apps, thus ensuring high grade application security.

4. Quixxi

   Quixxi is another such vulnerability scanner which scans for vulnerabilities in applications . It provides a platform for mobile application development that has all tools to keep mobile applications secure and run smoothly. It checks for vulnerabilities in applications which are either already developed or are in the process of development. With Quixxi you are free to work the way you want to, without being locked into a specific framework. You can add Quixxi to any IoT or mobile app which is finished or in development and within minutes you can protect data, monitor performance, identify and fix issues, manage licenses, aggregate information from multiple platforms, gather usage insights, connect with users, accelerate development and maximize revenue.

5. Ostorlab

   This is a cloud based application vulnerability scanner which works by uploading mobile application file. It aims at helping developers create Secure Mobile Apps.

Ostorlab's Security Scanner searches for security weaknesses and extracts critical information about the application behavior.

However, all these scanners generally use private knowledge sources as well as their own assessment techniques, and they do not provide any standardized and open means for describing and exchanging vulnerability descriptions within the community. Much of the work done in vulnerability analysis has defined the assessment infrastructure using its own vulnerability specification language arising compatibility and interoperability problems since the scanners so developed, cannot operate with other scanners, the output generated by them cannot be universally understood and also, the vulnerabilities which they detect cannot be traced at a universal or a standard level. Because of this, they fail to utilize the recorded information about a particular vulnerability and the patches or safety recommendations associated with them.

2.0.2 Other SCAP compliant tools

As discussed in the above chapter, SCAP was a solution to this problem, which standardized the vulnerability, compliance detection process and generating a standard output which can be universally understood by every other SCAP compliant product. The SCAP Validation Program tests the ability of products to employ SCAP standards. The NIST National Voluntary Laboratory Accreditation Program (NVLAP) accredits independent laboratories under the program to perform SCAP validations. A vendor seeking validation of a product can contact an NVLAP accredited SCAP validation laboratory for assistance in the validation process.

SCAP compliant products were developed by many commercial vendors for different operating systems- predominantly for the Microsoft windows family and red hat enterprise Linux family (RHEL). Below are a few examples of SCAP compliant products as men-

tioned on the official NIST website (NIST) for different platforms which harness some of the sub standards which come under the umbrella of SCAP standards :

1. OpenSCAP by RedHat : Platforms supported :
   - Red Hat Enterprise Linux 6.8 Client, 32 bit (x86)
   - Red Hat Enterprise Linux 6.8 Client, 32 bit (x64)
   - Red Hat Enterprise Linux 7.2 Client, 64 bit (x64))

   OpenSCAP is an auditing tool that utilizes the Extensible

2. SCAP Extensions for Microsoft System Center Configuration Manager 3.0 by Microsoft Platforms specified :
   - Microsoft Windows 7, 64 bit
   - Microsoft Windows 7, 32 bit

   The SCAP Extensions for Microsoft System Center

3. Policy Auditor 6.2 by Intel security : Platforms supported :
   - Microsoft Windows 7, 64 bit
   - Microsoft Windows 7, 32 bit
   - Microsoft Windows Vista, SP2
   - Microsoft Windows XP Pro, SP3
   - Red Hat Enterprise Linux 5.9 Desktop, 64 bit (x86 64)
   - Red Hat Enterprise Linux 5.9 Desktop, 32 bit (x86)

2.0.3 Attempt to detect vulnerabilities in android by using SCAP compliant tools

In a research paper Increasing Android Security using a Lightweight OVAL-based Vulnerability Assessment Framework (Festor *et al.*) by Martn Barrere, Gaetan Hurel, Remi Badonnel and Olivier Festor,the authors have instrumented their approach with an

experimental OVAL extension for Android . Their execution helps to specify known vulnerabilities for Android in a machine-readable manner and at the same time, with the view of standardizing the entire vulnerability detection and report generation process. They have tried to articulate the available android vulnerabilities description into an OVAL definition format and have provided a sample implementation of the entire process Ovaldroid (Ere *et al.*). Ovaldroid, the reference implementation of the Android Vulnerability Assessment tool, is an Android application completely written in Java. By means of a web service, Ovaldroid agents running on the Android platform connect to a remote database server in order to get new OVAL vulnerability definitions, perform local self-assessment activities, and report the obtained results. This experiment of ovaldroid was done in the year 2012. However, the official android schemas were released by mitre.org (by mitre.org) in the year 2014. So, the experiment was not carried out by using the android schemas as given by the mitre.org. Also, there has been no scanner which is SCAP compliant and scans for vulnerabilities which have been reported in the national vulnerability database in the recent years. I have attempted to build a preliminary vulnerability scanner which is in the form of an OVAL interpreter (and thus, SCAP compliant) , which scans for vulnerabilities reported because of the telephony features in android, in the years 2016 as given in the national vulnerability database (NVD). Since the scanner is SCAP compliant, extending it to include scanning for other vulnerabilities will be extremely simple since that would mean, adding on other definitions in the OVAL schema and the scanning report thus generated will be such that it can be understood by any other SCAP compliant tool. Thus this scanner is interoperable with all other SCAP compliant tools

Chapter 3

THEORY BEHIND EXECUTION

As discussed previously, OVAL (Open vulnerability and assessment language) promotes standardized vulnerability and configuration assessment and provides consistent and reproducible information assurance metrics. OVAL helps you to determine which vulnerabilities or configuration issues exist on your system. You may then use this information to obtain appropriate software patches and fix information for remediation. Hence, OVAL acts as a preventive measure. As an addition, you can use a vulnerability database like NVD (National vulnerability database),which includes information about vulnerabilities that OVAL does not, such as the severity of the problem, whether it is locally or remotely exploitable, remediation information, etc.

3.1 Components of OVAL

3.1.1 OVAL language

OVAL uses Extensible Markup Language (XML) because of its data centric approach which makes it easier to extract data and is machine readable. The main purpose of the OVAL Language is to standardize the three key steps of the assessment process,

- Representing configuration information of systems for testing
- Analyzing the system for the presence of the specified machine state

- Reporting the results of this assessment

The OVAL community has developed schemas approved by the OVAL Board written in Extensible Markup Language (XML) to serve as the framework and vocabulary of the OVAL Language. An OVAL System Characteristics schema represents system information; an OVAL Definition schema expresses an ideal specific machine state, and an OVAL Results schema reports the results of the assessment performed.

1. OVAL definition:

   OVAL Definitions are machine readable files written in XML to highlight any system vulnerability, configuration issue, programs and patches present on the system. For these assessments, we have different definition files which give us a roadmap.

   - OVAL Vulnerability Definitions - Tests that determine the presence of vulnerabilities on systems.

   - OVAL Compliance Definitions - Tests that determine whether the configuration settings of a system meet a security policy.

   - OVAL Inventory Definitions - Tests that determine whether a specific piece of software is installed on the system.

   - OVAL Patch Definitions - Tests that determine whether a particular patch is appropriate for a system.

   - OVAL Miscellaneous Definitions - Tests that do not fall into any of the four main classes.

2. OVAL System Characteristics:

   This file is generated by the OVAL Interpreter. It contains general information about the system from which data was collected, including the information that can be used to identify the system. All the objects that have been collected for assessment and their status are mentioned in this. After the system and object is mapped, the current

state of that object is specified for analysis.

3. OVAL results schema:

   OVAL Results schema stores the results of the evaluations performed on the system against the standard definitions. The current state of the machine is compared to the OVAL Definitions to check for vulnerabilities or issues. The data generated can be interpreted to take the necessary actions to mitigate the vulnerabilities. OVAL Results file can be of two types - full and thin. A 'thin' file means only a minimal amount of information is provided about the assessment of objects whereas a full file means that very detailed information is provided allowing in-depth reports to be generated from the results.

3.1.2 OVAL Repository

The OVAL Repository is the central meeting place for the OVAL Community to discuss, analyze, store, and disseminate OVAL definitions. The OVAL Language and any resulting OVAL content based upon the language that is stored in the OVAL Repository is free to use by any organization or individual for any research, development, and commercial purposes. The OVAL Repository Moderator evaluates and reviews definitions for publication in the OVAL Repository. Once new definitions are published in the OVAL Repository, they are subjected to community review. The OVAL Repository uses the publicly known vulnerabilities which describe a list of vulnerabilities and exposures. Hence, the OVAL Repository plays a very important role in maintaining OVAL as an international, information security, community effort.

3.1.3 OVAL Interpreter

The OVAL Interpreter evaluates OVAL Definitions. Based on a set of XML Definitions the Interpreter collects system information, evaluates it, and generates a detailed OVAL Results file. When we run the Interpreter, it will provide a list of OVAL Definition IDs and their results on the system. The working of an OVAL interpreter is shown in figure 3.1.

3.2 Working of OVAL

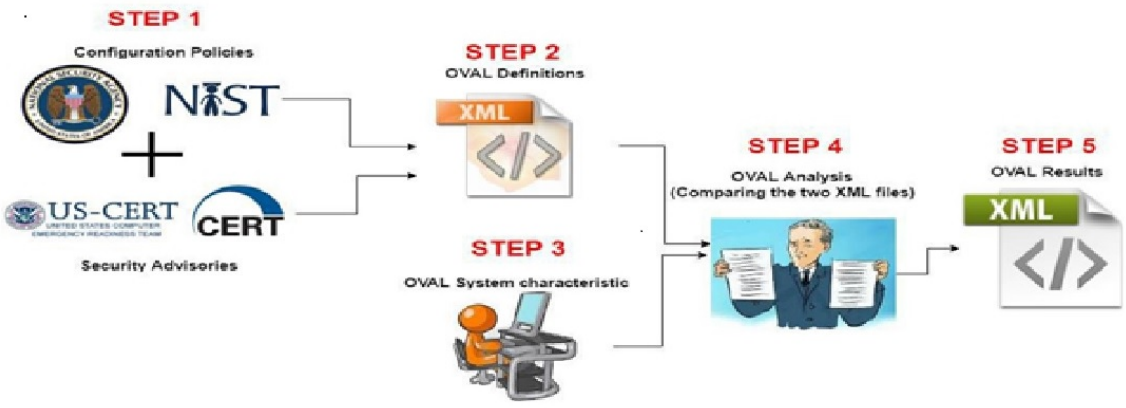


FIG. 3.1. Working of OVAL

1. Step 1

   - Configuration policies: Certain government agencies such as NSA (national security agency) and NIST (National institute of standards and technologies) develop Best Practices policy. Following these policies ensures system security.

   - Security advisories: CERT-CC (computer emergency response team co-ordination center), US-CERT (United States computer emergency readiness team), and other

organizations publish security advisories. These advisories warn of current threats and system vulnerabilities.

These best practices and advisories are used to create an OVAL definition file for a particular system.

2. Step 2

   Specific machine configuration details from the security advisories and configuration policy documents are extracted and encoded as an OVAL Definition. The OVAL Definition schema is used to define the XML framework for writing

3. Step 3

   The OVAL system characteristics file basically encodes the details of the system to be evaluated. The OVAL System Characteristics schema defines a standard XML format for representing system configuration information, which includes operating system parameters, installed software application settings, and other security relevant configuration values. The schema provides a list of system characteristics against which OVAL definitions can be compared in order to analyze a system for the presence of a particular machine state. By utilizing the provided OVAL System Characteristics file, other applications would not need to perform data collection, but rather can use the provided information to perform analysis.

4. Step 4

   The OVAL Definitions from Step 2 and the System Characteristics from Step 3 are compared to determine if the current system state is vulnerable or not vulnerable, by using an OVAL interpreter.

5. Step 5

   As mentioned in the Step 4, based on the set of definitions, the interpreter collects

system information, evaluates it, compares it with the OVAL definition and generates detailed OVAL Results file. Results of analysis are formatted as an OVAL Results document.

3.3 How to write an OVAL definition file

To take advantage of the interoperability enabled by using a standard language like OVAL (Open Vulnerability and Assessment Language), someone has to generate the initial definitions to be used. An OVAL Definition is a bunch of expected system details arranged in a standard, pre-defined way, such that the tool consuming this data knows how to use it. An OVAL Definition can be written to describe a computer vulnerability, a policy that one must comply with, a system patch, or any other specific machine state. How to write an OVAL definition file from scratch has been mentioned in the document (Oval.mitre.org)

1. Step 1: A New OVAL Definition XML File :

   Before a new OVAL Definition can be written, an XML file must be set up which will contain the OVAL definitions. This is shown in Figure 3.2. This XML file can be used to hold multiple new OVAL Definitions. The root element (the first XML tag, located at the top of the file) should be- ”oval_definitions”. The root element is also where each needed namespace is declared. Each component schema (independent, Windows, UNIX, Solairs, etc) used by the OVAL Definition being written, must have its corresponding namespace declared in the root element.

```
<oval_definitions xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5" xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5" xmlns:oval-def="
        http://oval.mitre.org/XMLSchema/oval-definitions-5" xmlns:ind-def="http://oval.mitre.org/XMLSchema/oval-definitions-5#independent"
    xmlns:win-def="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-common-5 oval-common-schema.xsd
        http://oval.mitre.org/XMLSchema/oval-definitions-5 oval-definitions-schema.xsd http://oval.mitre.org/XMLSchema/
        oval-definitions-5#independent independent-definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5#windows
        windows-definitions-schema.xsd">

</oval_definitions>
```

FIG. 3.2. Creating a file with OVAL definition

Each XML file must also have a ”generator” element (as shown in Figure 3.3), that holds information about when the file was compiled and what version of the OVAL Schema was used. The optional ”product _name” and ”product _version” child elements describe the name and version of the application used to generate the file. The required ”schema _version” child element holds the version of the OVAL Schema that the definition XML file has been written in and that should be used for validation. The required ”timestamp” child element holds information about when the particular OVAL document was compiled. The format for the timestamp is yyyy-mm-ddThh:mm:ss.The information contained in the ”generator” element does not specify when a definition (or set of definitions) was created or modified but rather when the actual XML file that contains the definition(s) was created. For example, the document might have pulled a bunch of existing OVAL Definitions together, each of the definitions having been created at some point in the past. The information in this case would be about when the combined document was created.

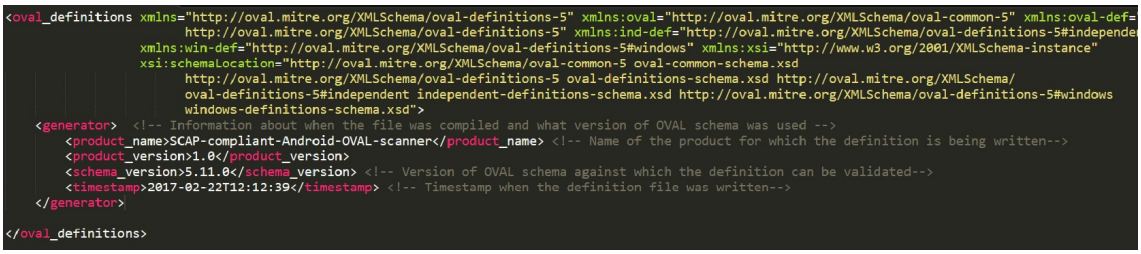


FIG. 3.3. Generator

2. Step 2 – Adding A Definition

   Once the XML file has been set up, we now must add an OVAL Definition to it. Each definition should coincide with a complete statement about the state of a machine. Individual definitions are grouped together as children of the ”definitions” element. Each individual definition has three required attributes: id, version, and class. These

are described in more detail below. A definition is separated into three sections: a metadata section, a notes section, and a criteria section. An example of definition is shown in Figure 3.4.

```
<oval_definitions xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5" xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5" xmlns:oval-def="
                  http://oval.mitre.org/XMLSchema/oval-definitions-5" xmlns:ind-def="http://oval.mitre.org/XMLSchema/oval-definitions-5#independent"
                xmlns:win-def="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-common-5 oval-common-schema.xsd
                     http://oval.mitre.org/XMLSchema/oval-definitions-5 oval-definitions-schema.xsd http://oval.mitre.org/XMLSchema/
                     oval-definitions-5#independent independent-definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5#windows
                     windows-definitions-schema.xsd">
    <generator>  <!-- Information about when the file was compiled and what version of OVAL schema was used -->
        <product_name>SCAP-compliant-Android-OVAL-scanner</product_name> <!-- Name of the product for which the definition is being written-->
        <product_version>1.0</product_version>
        <schema_version>5.11.0</schema_version> <!-- Version of OVAL schema against which the definition can be validated-->
        <timestamp>2017-02-22T12:12:39</timestamp> <!-- Timestamp when the definition file was written-->
    </generator>
    <definitions>  <!--collection of definition which provide a means to specify what endpoint information should be checked and what corresponding values
    are expected to be found.-->
        <definition id="oval:org.mitre.oval:def:101" version="1" class="vulnerability"> <!--required attributes. for id, org.mitre.oval namespace should be
        used. Note : this is a temporary ID. When submitted to the OVAL community,new randomly generated id is assigned from the OVAL repository. A
        definition is assigned Class value can be "vulnerability","compliance" ,"inventory","patch","Miscellaneous" -->
            <metadata>
            ......
            </metadata>

            <notes>
            ......
            </notes>

            <criteria>
            ......
            </criteria>
        </definition>

    </definitions>

</oval_definitions>
```

FIG. 3.4. Adding definitions

Assigning an ID :

For OVAL Definitions submitted to the OVAL repository, they should use the org.mitre.oval id namespace. New ids are assigned randomly from a pool that is managed by the OVAL Repository. When submitting new content to the OVAL repository, all new items (definitions, test, objects, states, and variables) should be assigned temporary ids. Once a new submission is reviewed and imported into the OVAL Repository official ids will be assigned. Temporary ids should be created in a namespace other than the org.mitre.oval id namespace.

The Version of a Definition:

The required version attribute holds the current version of the definition. Versions are integers, starting at 1 and incrementing every time a definition is modified.

What class does an OVAL Definition fall into?:

Each definition must be assigned a class to help group the definition by the type of system state it is describing. This helps find and sort definitions when the need arises. As mentioned earlier, there are five different classes to choose from: compliance, inventory, patch, vulnerability, and miscellaneous.

3. Step 3 – Filling In The Metadata : Each OVAL Definition should have some metadata associated with it to help identify it amongst an enormous collection of definitions.This is shown in Figure 3.5.

```
<metadata>
    <title>checking if the system is vulnerable to the "Year 2038 problem"</title>
    <affected family="Android">  <!--platforms affected by this vulnerability-->
        <platform>Android 4.x before 4.4.4</platform>
        <platform>Android 5.0.x before 5.0.2</platform>
        <platform>Android 5.1.x before 5.1.1</platform>
        <platform>Android 6.x before 2016-08-01</platform>
    </affected>
    <reference source="CVE" ref_id="CVE-2016-3831" ref_url="https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-3831"/> <!--Reference of
    where the vulnerability details can be found-->
    <description>
        The telephony component in the above mentioned versions of android, allow remote attackers to cause a denial of service (device crash)
        through a NITZ time value of 2038-01-19 or later that is mishandled by the system clock. This is known as a "Year 2038 problem" :  an
        issue for computing and data storage situations in which time values are stored or calculated as a signed 32-bit integer, and this
        number is interpreted as the number of seconds since 00:00:00 UTC on 1 January 1970 (the epoch)
    </description>
</metadata>
```

FIG. 3.5. Adding metadata

The Definition Title :

The required ”title” child element holds a short string that is used to quickly identify the definition.

The ”affected” element:

The affected metadata item contains information about the system(s) for which the definition has been written. This is just metadata and not part of the criteria.

References:

References can be added to the metadata section of an OVAL Definition to link it to a definitive external name. For example, a vulnerability definition can include a reference to the CVE Identifier that represents the vulnerability being defined. Each

reference element found in an OVAL Definition can include an external ID (for example the CVE ID) and a URL for finding information about the external ID.

The Definition Description:

The description of the OVAL definition will contain the description of the vulnerability contained in the CVE entry. In the case of vulnerability definitions without a CVE identifier, compliance or patch definitions the description should adequately convey the purpose of the definition. It should either describe the vulnerability in question, the patch and its function or what constitutes compliance. This section is for human consumption so it is left to the definition writers discretion to determine relevant content.

4. Step 4 – Notes

   Each definition can have optional notes associated with it. Each note contains some information about the definition or about the tests that the definition references. A note may record an unresolved question about the definition or test or present the reason as to why a particular approach was taken. To add a note, simply add a ”note” element as a child of the ”notes” element.An example of this section is shown in Figure 3.6.

```
<notes>
    <note>
        We will need to check whether telephony exists on the system and whether the operating system is of the
        "affected" section. If both tests return true, then the vulnerability exists on the system.
    </note>
</notes>
```

FIG. 3.6. Adding notes

5. Step 5 – Creating the Criteria

   Once the metadata and any notes have been added to the definition, it is time to start building the criteria. The criteria portion of an OVAL Definition is the crux of what

is being described. It joins individual tests together with a logical operator to specify the specific system state.To get started, a ”criteria” element is added as a child of the ”definition” element. Inside each ”criteria” element can be found other ”criteria” elements, ”criterion” elements, and ”extended _definition” elements. Each ”criteria” element has an operator attribute who’s value is either AND, OR, XOR, or ONE. This value determines how the result of the criteria statement will be determined. If the operator is AND, then every test referenced by the criteria must be true for the entire criteria to return true. If the operator is OR, then at least one test referenced by the criteria must be true for the entire criteria to return true. The criteria is a recursive element. That means you can have new ”criteria” elements inside existing ”criteria” elements. The reason for this is to enable nesting of logical statements. The ability to nest these criteria statements allows complex logic to be executed. An example of this is shown in Figure 3.7.

```
<criteria operator="AND"> <!--this is a recursive element which can contain one or more criterias which in turn can contain criterion. The
value of the 'operator' determines the final result of the test by performing the corresponding logical operation on the results of the
individual tests-->
    <criterion test_ref= "oval:org.mitre.oval:tst:101" comment="Telephony exists on the operating system" />
    <!--<criterion test_ref= "oval:org.mitre.oval:tst:102" comment="The version of the operating system is one of the versions mentioned in
    the 'affected platforms' section"/> -->
    <criteria operator="OR">
        <criterion test_ref= "oval:org.mitre.oval:tst:102" comment="The version of the operating system is one of the versions mentioned in
        the 'affected platforms' section is 4.0"/>
        <criterion test_ref= "oval:org.mitre.oval:tst:103" comment="The version of the operating system is one of the versions mentioned in
        the 'affected platforms' section is 4.1,4.2,4.3"/>
        <criterion test_ref= "oval:org.mitre.oval:tst:104" comment="The version of the operating system is one of the versions mentioned in
        the 'affected platforms' section is 5.x"/>
        <criterion test_ref= "oval:org.mitre.oval:tst:105" comment="The version of the operating system is one of the versions mentioned in
        the 'affected platforms' section is 6.x"/>
    </criteria>
</criteria>
```

FIG. 3.7. Adding criteria

6. Step 6 – Adding A Test

   Once the criteria of the OVAL Definition has been created, the next step is to add the associated tests. You first need to assign an ID to the test which will uniquely identify the test. The optional check _existence attribute determines how many objects in the specified set must exists for the test to be true. For example, if a value of ’all exist’ is

given, every object defined by the OVAL Object must exist on the system for the test to return true. If the OVAL Object uses a variable reference, then every value of that variable must exist. Note that a pattern match defines a set of matching objects found on a system. So when check _existence = 'all _exist' and a regex matches anything on a system the test will evaluate to true. (since all matching objects on the system were found on the system). When check _existence = 'all _exist' and a regex does not match anything on a system the test will evaluate to false.The required check attribute determines how many of the existing objects must satisfy the state requirements. (For example: Should the test check that all files match a specified version or that at least one file matches the specified version?)

```
<tests>
    <telephony_test id="oval:org.mitre.oval:tst:101" check="" check_existence="" comment="Test to check if telephony exists on the operating system">
        <object object_ref="oval:org.mitre.oval:obj:101"></object>
        <state state_ref="oval:org.mitre.oval:ste:101"></state>
    </telephony_test>
    <systemdetails_test id="oval:org.mitre.oval:tst:102" check="" check_existence="" comment="Test to know the version of OS">
        <object object_ref="oval:org.mitre.oval:obj:201"></object>
        <state state_ref="oval:org.mitre.oval:ste:201"></state>
    </systemdetails_test>
    <systemdetails_test id="oval:org.mitre.oval:tst:103" check="" check_existence="" comment="Test to know the version of OS">
        <object object_ref="oval:org.mitre.oval:obj:201"></object>
        <state state_ref="oval:org.mitre.oval:ste:202"></state>
    </systemdetails_test>
    <systemdetails_test id="oval:org.mitre.oval:tst:104" check="" check_existence="" comment="Test to know the version of OS">
        <object object_ref="oval:org.mitre.oval:obj:201"></object>
        <state state_ref="oval:org.mitre.oval:ste:203"></state>
    </systemdetails_test>
    <systemdetails_test id="oval:org.mitre.oval:tst:105" check="" check_existence="" comment="Test to know the version of OS">
        <object object_ref="oval:org.mitre.oval:obj:201"></object>
        <state state_ref="oval:org.mitre.oval:ste:204"></state>
    </systemdetails_test>
    <systemdetails_test id="oval:org.mitre.oval:tst:106" check="" check_existence="" comment="Test to know the version of OS">
        <object object_ref="oval:org.mitre.oval:obj:201"></object>
        <state state_ref="oval:org.mitre.oval:ste:205"></state>
    </systemdetails_test>
</tests>
```

FIG. 3.8. Adding tests

7. Step 7 – Adding Necessary Objects

   This is shown in Figure 3.9.

8. Step 8 – Adding Necessary States

   This is shown in Figure 3.10.

9. Step 9 – Submitting The OVAL Definition

```
<objects>
    <telephony_object id="oval:org.mitre.oval:obj:101" version="1" comment="telephonyExists">
    </telephony_object>
    <systemdetails_object id="oval:org.mitre.oval:obj:201" version="1" comment="knowing the OS">
    </systemdetails_object>
</objects>
```

FIG. 3.9. Adding objects

```
<states>
    <telephony_state id="oval:org.mitre.oval:ste:101" version="1"> <!--Checking the network type and the country iso code to check if telephony exists
    on the system-->
        <network_type datatype="string" operation="equals" entity_check="all">LTE</network_type>
        <sim_country_iso datatype="int" operation="equals" entity_check="all">1</sim_country_iso>
    </telephony_state>
    <systemdetails_state id="oval:org.mitre.oval:ste:201" version="1"> <!--checking the version of the OS on android Operating system-->
        <os_version_codename datatype="string" operation="equals">Ice cream sandwich</os_version_codename>
        <!--code names : Cupcake (v1.5) Donut (v1.6),Eclair (v2.0),FroYo (v2.2),Gingerbread (v2.3),Honeycomb (v3.0), Ice Cream Sandwich (v4.0), Jelly
        Bean (v4.1, v4.2, v4.3), KitKat (v4.4) Lollipop (v5.0),Marshmallow (v6.0),Nougat (v7.0)-->
    </systemdetails_state>
    <systemdetails_state id="oval:org.mitre.oval:ste:202" version="1"> <!--checking the version of the OS on android Operating system-->
        <os_version_codename datatype="string" operation="equals">Jelly Beans</os_version_codename>
        <!--code names : Cupcake (v1.5) Donut (v1.6),Eclair (v2.0),FroYo (v2.2),Gingerbread (v2.3),Honeycomb (v3.0), Ice Cream Sandwich (v4.0), Jelly
        Bean (v4.1, v4.2, v4.3), KitKat (v4.4) Lollipop (v5.0),Marshmallow (v6.0),Nougat (v7.0)-->
    </systemdetails_state>
    <systemdetails_state id="oval:org.mitre.oval:ste:203" version="1"> <!--checking the version of the OS on android Operating system-->
        <os_version_codename datatype="string" operation="equals">Lollipop</os_version_codename>
        <!--code names : Cupcake (v1.5) Donut (v1.6),Eclair (v2.0),FroYo (v2.2),Gingerbread (v2.3),Honeycomb (v3.0), Ice Cream Sandwich (v4.0), Jelly
        Bean (v4.1, v4.2, v4.3), KitKat (v4.4) Lollipop (v5.0),Marshmallow (v6.0),Nougat (v7.0)-->
    </systemdetails_state>
    <systemdetails_state id="oval:org.mitre.oval:ste:204" version="1"> <!--checking the version of the OS on android Operating system-->
        <os_version_codename datatype="string" operation="equals">Marshmallow</os_version_codename>
        <!--code names : Cupcake (v1.5) Donut (v1.6),Eclair (v2.0),FroYo (v2.2),Gingerbread (v2.3),Honeycomb (v3.0), Ice Cream Sandwich (v4.0), Jelly
        Bean (v4.1, v4.2, v4.3), KitKat (v4.4) Lollipop (v5.0),Marshmallow (v6.0),Nougat (v7.0)-->
    </systemdetails_state>
    <systemdetails_state id="oval:org.mitre.oval:ste:205" version="1"> <!--checking the version of the OS on android Operating system-->
        <os_version_codename datatype="string" operation="equals">Nougat</os_version_codename>
        <!--code names : Cupcake (v1.5) Donut (v1.6),Eclair (v2.0),FroYo (v2.2),Gingerbread (v2.3),Honeycomb (v3.0), Ice Cream Sandwich (v4.0), Jelly
        Bean (v4.1, v4.2, v4.3), KitKat (v4.4) Lollipop (v5.0),Marshmallow (v6.0),Nougat (v7.0)-->
    </systemdetails_state>
</states>
```

FIG. 3.10. Adding states

If desired, submissions of OVAL vulnerability, compliance, inventory, and patch definitions may be emailed to MITRE in the proper format by members of the security community who have registered for the Community Forum, or by OVAL Board members. All submissions will be reviewed by the OVAL content team and the OVAL Editor prior to being posted on the OVAL Web site for comment and public debate.

3.4 The OVAL system characteristics file

This is an XML encoding of the details of a system file - versions running, processes and patches installed etc. It provides a snapshot of the system which can be saved for auditing purposes and can be used for analysis of the system. This file drives the OVAL

evaluation and should be generated by the OVAL interpreter. It gives the state of the system which can then be evaluated against the expected system state mentioned in the definition file. The criteria in the definition file can thus be evaluated to be true or false , using this information encoded in the system characteristics file. The system characteristics file consists of the following section :

- Generator :

This consists of the information about how the OVAL Document was created and information about product name, product version, schema version, timestamp. It is not about the content, but about the document

- System info section :

Identifies the system from which the set of data was collected. It includes information about primary host name, Operating System name and version Interfaces. See Figure 3.11.

```
<system_info>
   <os_name>Microsoft Windows XP Professional Service Pack 2</os_name>
   <os_version>5.1.2600</os_version>
   <architecture>INTEL32</architecture>
   <primary_host_name>MyHostName.example.com</primary_host_name>
   <interfaces>
     <interface>
       <interface_name>Broadcom NetXtreme ...</interface_name>
       <ip_address>192.68.0.10</ip_address>
       <mac_address>00-11-22-AA-BB-33</mac_address>
     </interface>
   </interfaces>
 </system_info>
```

FIG. 3.11. System info section

- Collected objects section :

This section provides a mapping from the objects specified by an OVAL Definition to the set of items collected on a host. This is an optional section . If the collected objects section

is missing in the system characteristics file, then the entire file has to be searched for the valid information to be used for evaluation of a particular criteria for a particular definition in the OVAL definition file. See Figure 3.12.

```
<collected_objects>
  ...
  <object flag="complete" id="oval:org.mitre.oval:obj:281" version="1">
    <reference item_ref="2"/>
  </object>
  <object flag="complete" id="oval:org.mitre.oval:obj:38" version="1">
    <variable_value variable_id="oval:org.mitre.oval:var:200">C:\Program
    Files</variable_value>
    <reference item_ref="3"/>
  </object>
  ...
</collected_objects>
```

FIG. 3.12. collected objects section

- System data section :

This section enlists set of items found on the host system. This includes the actual data like reg keys, file attrs, permissions, etc. See Figure 3.13.

3.5 The OVAL result file

The XML encoding of the results of an analysis include the information about which systems are vulnerable, which systems are non-compliant, and which patches should be installed and also include the details - why are they vulnerable, why are they non-compliant, why should a patch be installed. There are two types of result files full and thin.

- A value of 'thin' means only a minimal amount of information is provided as shown in Figure 3.14.

- A value of 'full' means that very detailed information is provided allowing in-depth reports to be generated from the results as shown in Figure 3.15.

```
<system_data>
    ...
    <file_item id="3" xmlns="http://oval.mitre.org/XMLSchema/oval-system-characteristics-5#windows">
      <path>C:\Program Files\Common Files\Microsoft Shared\WMI</path>
      <filename>WmiScriptUtils.dll</filename>
      <owner>Administrators</owner>
      <size datatype="int">54272</size>
      <a_time datatype="int">1183042181</a_time>
      <c_time datatype="int">1165044708</c_time>
      <m_time datatype="int">1165044708</m_time>
      <ms_checksum>60432</ms_checksum>
      <version datatype="version">8.0.50727.762</version>
      <type>FILE_TYPE_DISK</type>
      <development_class>SP</development_class>
      <company>Microsoft Corporation</company>
      <internal_name>WMIScriptUtils.DLL</internal_name>
      <language status="not collected"/>
      <original_filename>WMIScriptUtils.DLL</original_filename>
      <product_name>Microsoft® Visual Studio® 2005</product_name>
      <product_version>8.0.50727.762</product_version>
    </file_item>
    ...
 </system_data>
```

FIG. 3.13. system data section

(Nvd.nist.gov) has been referred for the format of the OVAL results file. An OVAL result file consists of the following sections :

Generator:

It contains information about how a document was created. It includes information about the product name, product version, schema version, timestamp.

Directives (optional) :

This section reports about the contents of the results document. It mentions whether the results thus produced are in the full format or thin format, for every case. This has been shown in Figure 3.13.

OVAL definitions (optional) :

This provides us with an exact copy of the definitions evaluated. This is an optional field. When used along with full results, it allows for a complete snapshot of the evaluation results in one document.

Results

```
...
<results>
   <system>
    <definitions>
      <definition definition_id="" version="1" result="true">
      <definition definition_id="" version="1" result="true">
      <definition definition_id="" version="1" result="true">
      <definition definition_id="" version="1" result="true">
    </definitions>
   </system>
   <system>
   ...
   </system>
</results>
  ...
```

FIG. 3.14. Thin result file

This section depends upon whether the result to be generated is in the full or the thin format. Possible result attribute values are :

- True

- False

- Unknown

- Error

- Not evaluated

- Not applicable

The above information was used to write an OVAL definition file and then generate the results file in the appropriate format. This experiment was carried out for a certain set of vulnerabilities which were reported in the national vulnerability database during the years of 2016 , specifically for the feature telephony present in the android operating system.

```
...
 <results>
   <system>
     <definitions>
       <definition definition_id="" version="1" result="true">
         <criteria operator="AND" result="false">
           <criterion test_ref="" version="1" result="true"/>
           <criterion test_ref="" version="1" result="true"/>
         </criteria>
       </definition>
     </definitions>
     <tests>
       <test test_id="" version="2" check="at least one"
         result="true"> <tested_item item_id="1" result="true"/>
         <tested_item item_id="1" result="true"/>
         <tested_variable variable_id="">C:\WINDOWS</tested_variable>
       </test>
     </tests>
     <oval_system_characteristics>
   </system>
 </results>
 ...
```

FIG. 3.15. Full result file

3.6 The vulnerabilities

1. CVE-2016-3831 :

   The telephony component in Android 4.x before 4.4.4, 5.0.x before 5.0.2, 5.1.x before 5.1.1, and 6.x before 2016-08-01 allows remote attackers to cause a denial of service (device crash) via a NITZ time value of 2038-01-19 or later that is mishandled by the system clock. This is related to a famous problem known as Year 2038 problem.

   (The Year 2038 problem : The Year 2038 problem is an issue for computing and data storage situations in which time values are stored or calculated as a signed 32-bit integer, and this number is interpreted as the number of seconds since 00:00:00 UTC on 1 January 1970 (the epoch). Such implementations cannot encode times after

```
<directives>
    <definition_true reported="true" content="full"/>
    <definition_false reported="false"/>
    <definition_unknown reported="true" content="thin"/>
    <definition_error reported="false"/>
    <definition_not_evaluated reported="true" content="thin" />
    <definition_not_applicable reported="true" content="thin" />
</directives>
```

FIG. 3.16. Directives

03:14:07 UTC on 19 January 2038.)

2. CVE-2016-6771:

   An elevation of privilege vulnerability in Telephony could enable a local malicious application to access system functions beyond its access level. This issue is rated as Moderate because it is a local bypass of restrictions on a constrained process. Android versions affected by this vulnerability are 6.0, 6.0.1, 7.0.

3. CVE-2016-6763:

   An elevation of privilege vulnerability in Telephony could enable a local malicious application to access system functions beyond its access level. This issue is rated as Moderate because it is a local bypass of restrictions on a constrained process. Android versions affected by this vulnerability are 6.0, 6.0.1, 7.0.

4. CVE-2016-3922:

   libril/RilSapSocket.cpp in Telephony in Android 6.x before 2016-10-01 and 7.0 before 2016-10-01 relies on variable-length arrays, which allows attackers to gain privileges via a crafted application, as known as internal bug 30202619.

5. CVE-2016-3914:

Race condition in providers/telephony/MmsProvider.java in Telephony in Android versions 4.x before 4.4.4, 5.0.x before 5.0.2, 5.1.x before 5.1.1, 6.x before 2016-10-01, and 7.0 before 2016-10-01 allows attackers to gain privileges via a crafted application that modifies a database between two open operations, aka internal bug 30481342.

Chapter 4

EXECUTION OF THE INTERPRETER

4.1 The execution

4.1.1 Writing the OVAL definition for the vulnerabilities

As a first step towards writing the OVAL interpreter, as mentioned in the previous chapter and by following the guidelines given by the mitre.org official documentation, I encoded an OVAL_definition.xml file. The first task was to shortlist the vulnerabilities recorded and reported in the National Vulnerability Database(NVD) because of the telephony feature present in the android operating system which were reported in the year 2016. I studied the five vulnerabilities and traced the remediations and patches suggested for some of them as documented in the google bulletins. The conditions responsible for each of the vulnerabilities to exist on the android operating system were then enlisted and were used to write the appropriate criteria and the Boolean conditions in the definition xml file.

4.1.2 Scanning by using the OVAL interpretor

We now need the information of the system whose safety from the vulnerabilities has to be detected. We have the definition file which describes the tests to be carried out

by using the system information to determine whether the system is vulnerable to attacks from those particular vulnerabilities. So now, the acquired information has to be evaluated against the tests prescribed and appropriate standardized results have to be generated. To do so, at the implementation level, the following steps have been carried out :

1. Unmarshalling of the definition file

   The implementation has been done in JAVA by using the JAXB library. Java Architecture for XML Binding (JAXB) is a software framework that allows Java developers to map Java classes to XML representations and vice versa. JAXB provides two main features: the ability to marshal Java objects into XML and the inverse, i.e. to unmarshal XML back into Java objects. In other words, JAXB allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure. In the first case, I have the XML definition file which has to be unmarshalled into the appropriate JAVA classes. This has been achieved by writing the appropriate JAVA classes in accordance with the XML elements and attributes used in the definition file. Unmarshalling results in the conversion of XML elements and attributes in the form of classes and objects and this eases the execution of the mentioned tests in the criteria.

2. Executing a restful webservice

   The decision of executing a webservice was taken so as to avoid the entire XML parsing module to work on the android device since the dependent jars would increase the total size of the apk and also, some XML parsing tools do not work in the android development environment. Tomcat server has been used to deploy the restful webservice which executes the entire scanning process. This webservice can be hit from any android device by passing the following three parameters :

- OS type

- Telephony type

- The android OS details (for the generation of the system characteristics file)

These parameters are then used to generate a system characteristic file, compare and evaluate the results by using the oval_definition file and then generate a results.xml file which shows the results for the evaluation in a standard format.

3. Generation of the system characteristics file

   The system characteristics file provides a snapshot of the operating system in consideration, which, in this case, is the particular android device which is to be scanned for the specified vulnerabilities. This file is responsible for driving the OVAL evaluation. The system_data section in the system characteristics file is the one which involves all the necessary details about the operating system . It is the set of items and their values actually found on the host system.

   In the case of android operating system, a system characteristics file is generated in accordance with the oval_characteristics_schema.xsd which has been provided by the OVAL community, for android, and the system information received as a parameter to the webservice. The inbuilt packages in android, like TelephonyManager and Build are used to extract the required values in this case. The JAXB libraries have been used to encode this information and convert it into an xml file having appropriate elements and attributes , by marshalling the gathered system information.

4. Marshalling to generate the results.xml file

   The information from the systems_characteristics file about the specific android device is used to evaluate the specific tests in the oval definition file, which results in a Boolean value for every set of tests carried for each of the vulnerabilities and the results thus generated are encoded back into xml to generate the vulnerability scanning

result in the standardized form. To generate the results.xml for the scanning results, JAXB jar is used for the marshalling mechanism.

4.2 A sample use case

This use case scenario takes us through the steps to check for one of the vulnerabilities considered : CVE-3831 .

4.2.1 Writing the OVAL definition file

The image describes the criteria for the vulnerability CVE-3831 to exist on the android device. So for the vulnerability to exist on the system: Evaluate(Oval.org.mitre.oval:tst:101) AND (Evaluate(Oval.org.mitre.oval:tst:102) OR Evaluate(Oval.org.mitre.oval:tst:103) OR Evaluate(Oval.org.mitre.oval:tst:104) OR Evaluate(Oval.org.mitre.oval:tst:105)) = true. Which translates to : If telephony exists on the system and the operating system version is either 4.0,4.1,4.2,4.3,5.x or 6.x, then this vulnerability exists.

```
<criteria operator="AND"> <!--this is a recursive element which can contain one or more criterias which in turn can contain criterion. The
value of the 'operator' determines the final result of the test by performing the corresponding logical operation on the results of the
individual tests-->
    <criterion test_ref= "oval:org.mitre.oval:tst:101" comment="Telephony exists on the operating system" />
    <!--<criterion test_ref= "oval:org.mitre.oval:tst:102" comment="The version of the operating system is one of the versions mentioned in
    the 'affected platforms' section"/> -->
    <criteria operator="OR">
        <criterion test_ref= "oval:org.mitre.oval:tst:102" comment="The version of the operating system is one of the versions mentioned in
        the 'affected platforms' section is 4.0"/>
        <criterion test_ref= "oval:org.mitre.oval:tst:103" comment="The version of the operating system is one of the versions mentioned in
        the 'affected platforms' section is 4.1,4.2,4.3"/>
        <criterion test_ref= "oval:org.mitre.oval:tst:104" comment="The version of the operating system is one of the versions mentioned in
        the 'affected platforms' section is 5.x"/>
        <criterion test_ref= "oval:org.mitre.oval:tst:105" comment="The version of the operating system is one of the versions mentioned in
        the 'affected platforms' section is 6.x"/>
    </criteria>
</criteria>
```

FIG. 4.1. Criteria for the vulnerability CVE-3831

4.2.2 Running the scanner

The first step to run the scanner is running the webservice (as shown in Figure 4.2), which , when hit through a device, will execute the scanner. The second step involves

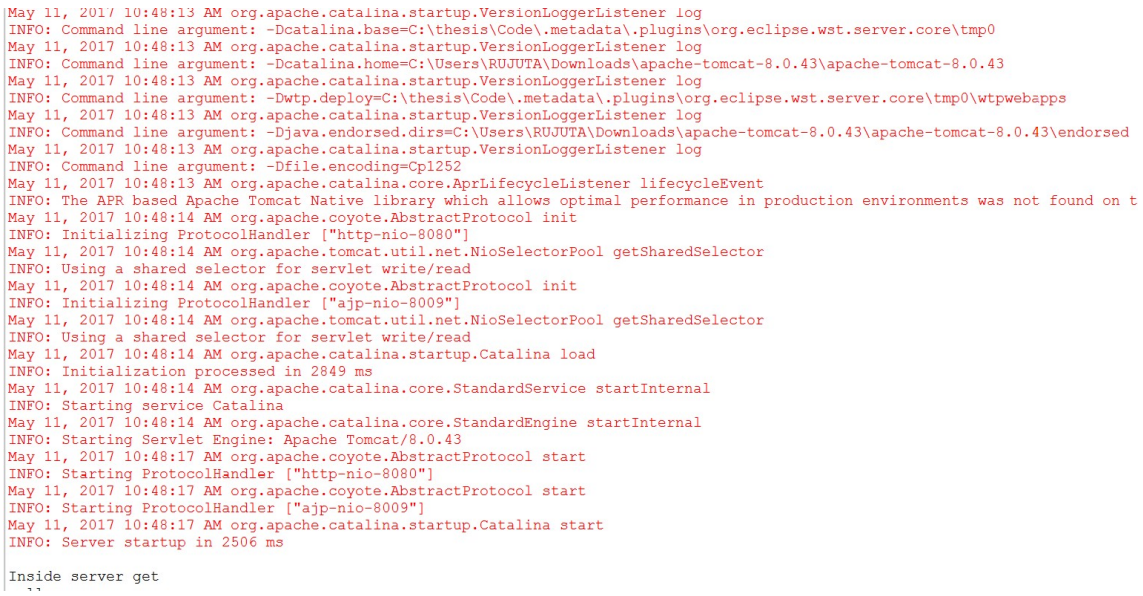


FIG. 4.2. Starting the server

hitting this web service by an android device for which the scanning must be carried out.The third step involves generating the system characteristics file by the vulnerability scanner.As the Figure 4.3 shows, the OS version of this particular device is 7.0 and telephony does exist on the system since the network_type() is LTE and not unknown. Therefore, the vulnerability SHOULD NOT EXIST (as the OS version in 7.0 while this vulnerability exists for 4.x,5.x,6.x).The fourth step which is internally carried out by the scanner is the evaluation of the two criteria as mentioned in the definition file by taking the appropriate values from the generated system_characteristics file and when done , then generating the scanning results in the form of standard OVAL results file. The highlighted part in the Figure 4.4 shows the evaluation of the test for CVE-3831 which results to false which means the vulnerability does not exist.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<oval_system_characteristics>
    <generator>
        <product_name>SCAP-compliant-Android-OVAL-scanner</product_name>
        <schema_version>5.11.0</schema_version>
        <timestamp>2017-02-22T12:12:39</timestamp>
        <product_version>1.0</product_version>
    </generator>
    <system_info>
        <os_name>Android</os_name>
        <os_version>7.0</os_version>
        <architecture>x86</architecture>
        <primary_host_name>fc3d75b473e5a22f</primary_host_name>
    </system_info>
    <system_data>
        <system_details_item>
            <hardware>ranchu</hardware>
            <manufacturer>unknown</manufacturer>
            <model>Android SDK built for x86</model>
            <product>sdk_google_phone_x86</product>
            <cpu_abi>x86</cpu_abi>
            <build_fingerprint>Android/sdk_google_phone_x86/generic_x86:7.0/NYC/3357259:userdebug/test-keys</build_fingerprint>
            <os_version_codename>REL</os_version_codename>
            <os_version_build_number>3357259</os_version_build_number>
            <os_version_release_name>7.0</os_version_release_name>
            <os_version_sdk_number>24</os_version_sdk_number>
            <hardware_keystore>abc</hardware_keystore>
        </system_details_item>
        <telephony_item id="1">
            <network_type>LTE</network_type>
            <sim_country_iso>us</sim_country_iso>
            <sim_operator_code>310260</sim_operator_code>
        </telephony_item>
    </system_data>
</oval_system_characteristics>
```

FIG. 4.3. The system characteristics file

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<oval_definitions>
    <generator>
        <product_name>SCAP-compliant-Android-OVAL-scanner</product_name>
        <schema_version>5.11.0</schema_version>
        <timestamp>2017-02-22T12:12:39</timestamp>
        <product_version>1.0</product_version>
    </generator>
    <directives>
        <definition_false reported="false"/>
        <definition_true reported="true" content="thin"/>
    </directives>
    <results>
        <definitions>
            <definition id="oval:org.mitre.oval:def:101" result="false"/>
            <definition id="oval:org.mitre.oval:def:201" result="true"/>
            <definition id="oval:org.mitre.oval:def:301" result="true"/>
            <definition id="oval:org.mitre.oval:def:401" result="true"/>
            <definition id="oval:org.mitre.oval:def:501" result="true"/>
        </definitions>
    </results>
</oval_definitions>
```

FIG. 4.4. The result file

Chapter 5

EVALUATION OF THE RESULTS

5.1 Case 1

Operating system : Gingerbread (v2.3) and telephony : exists.

Observation : The result evaluates to false as none of the vulnerabilities exist if the OS version is 2.3. (Refer Figure 5.1 and 5.2)

5.2 Case 2

Operating system : Lollipop (v5.0) and telephony : exists.

Observation : Only those test results evaluate to false whose criteria for the tests to pass does not include the OS version 5.0. The other tests evaluate to true. (Refer Figure 5.3 and 5.4)

5.3 Case 3

Operating system : Marshmallow (v6.0) and telephony : exists.

Observation : All test results evaluates to true as all the vulnerabilities exist if the OS version is 6.0. (Refer Figure 5.5 and 5.6)

5.4 Case 4

Operating system : Nougat (v7.0) and telephony : exists.

Observation : The last four test results evaluates to true as those vulnerabilities exist if the OS version is 7.0, whereas the first vulnerability evaluates to false. (Refer Figure 5.7 and 5.8)

5.5 Case 5

Operating system : Nougat (v7.0) and telephony : does not exist.

Observation : As telephony does not exists, all tests evaluate to false. (Refer Figure 5.9 and 5.10)

FIG. 5.1. Case 1: The system characteristics file

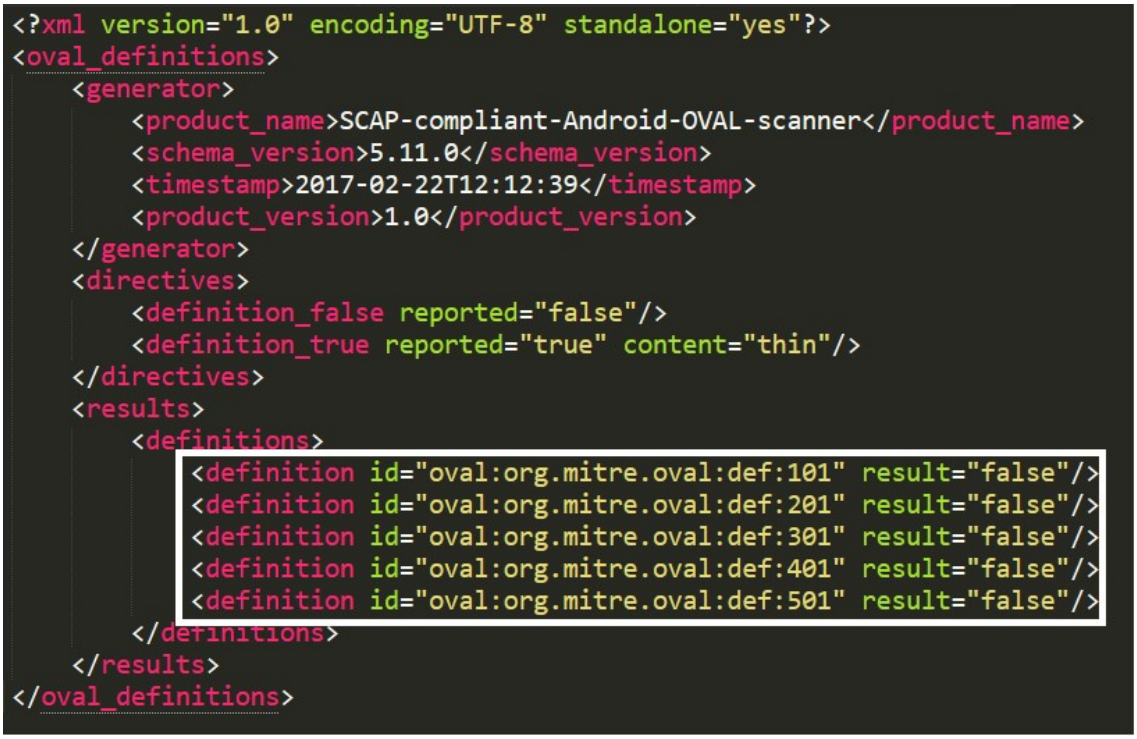
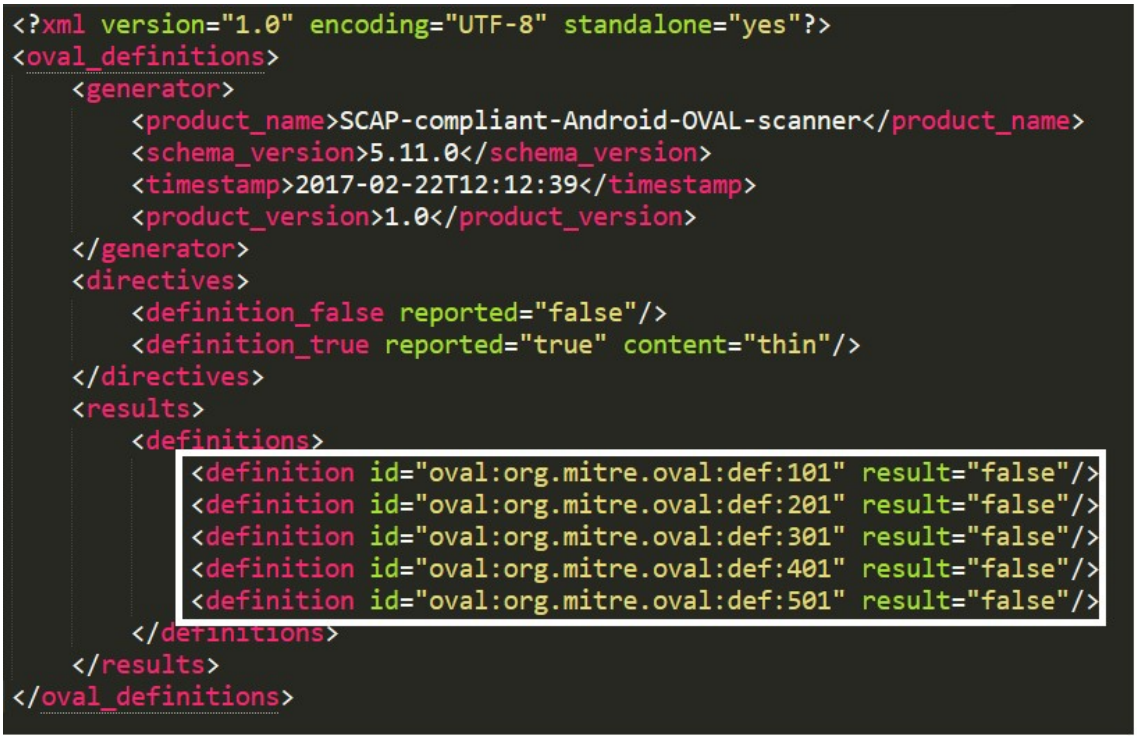


FIG. 5.2. Case 1: The result file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<oval_system_characteristics>
    <generator>
        <product_name>SCAP-compliant-Android-OVAL-scanner</product_name>
        <schema_version>5.11.0</schema_version>
        <timestamp>2017-02-22T12:12:39</timestamp>
        <product_version>1.0</product_version>
    </generator>
    <system_info>
        <os_name>Android</os_name>
        <os_version>5.0</os_version>
        <architecture>x86</architecture>
        <primary_host_name>fc3d75b473e5a22f</primary_host_name>
    </system_info>
    <system_data>
        <system_details_item>
            <hardware>ranchu</hardware>
            <manufacturer>unknown</manufacturer>
            <model>Android SDK built for x86</model>
            <product>sdk_google_phone_x86</product>
            <cpu_abi>x86</cpu_abi>
            <build_fingerprint>Android/sdk_google_phone_x86/generic_x86:7.0/NYC/3357259:userdebug/test-keys</build_fingerprint>
            <os_version_codename>REL</os_version_codename>
            <os_version_build_number>3357259</os_version_build_number>
            <os_version_release_name>5.0</os_version_release_name>
            <os_version_sdk_number>24</os_version_sdk_number>
            <hardware_keystore>abc</hardware_keystore>
        </system_details_item>
        <telephony_item id="1">
            <network_type>LTE</network_type>
            <sim_country_iso>us</sim_country_iso>
            <sim_operator_code>310260</sim_operator_code>
        </telephony_item>
    </system_data>
</oval_system_characteristics>
```

FIG. 5.3. Case 2: The system characteristics file



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<oval_definitions>
    <generator>
        <product_name>SCAP-compliant-Android-OVAL-scanner</product_name>
        <schema_version>5.11.0</schema_version>
        <timestamp>2017-02-22T12:12:39</timestamp>
        <product_version>1.0</product_version>
    </generator>
    <directives>
        <definition_false reported="false"/>
        <definition_true reported="true" content="thin"/>
    </directives>
    <results>
        <definitions>
            <definition id="oval:org.mitre.oval:def:101" result="false"/>
            <definition id="oval:org.mitre.oval:def:201" result="false"/>
            <definition id="oval:org.mitre.oval:def:301" result="false"/>
            <definition id="oval:org.mitre.oval:def:401" result="false"/>
            <definition id="oval:org.mitre.oval:def:501" result="false"/>
        </definitions>
    </results>
</oval_definitions>
```

FIG. 5.4. Case 2: The result file

FIG. 5.5. Case 3: The system characteristics file

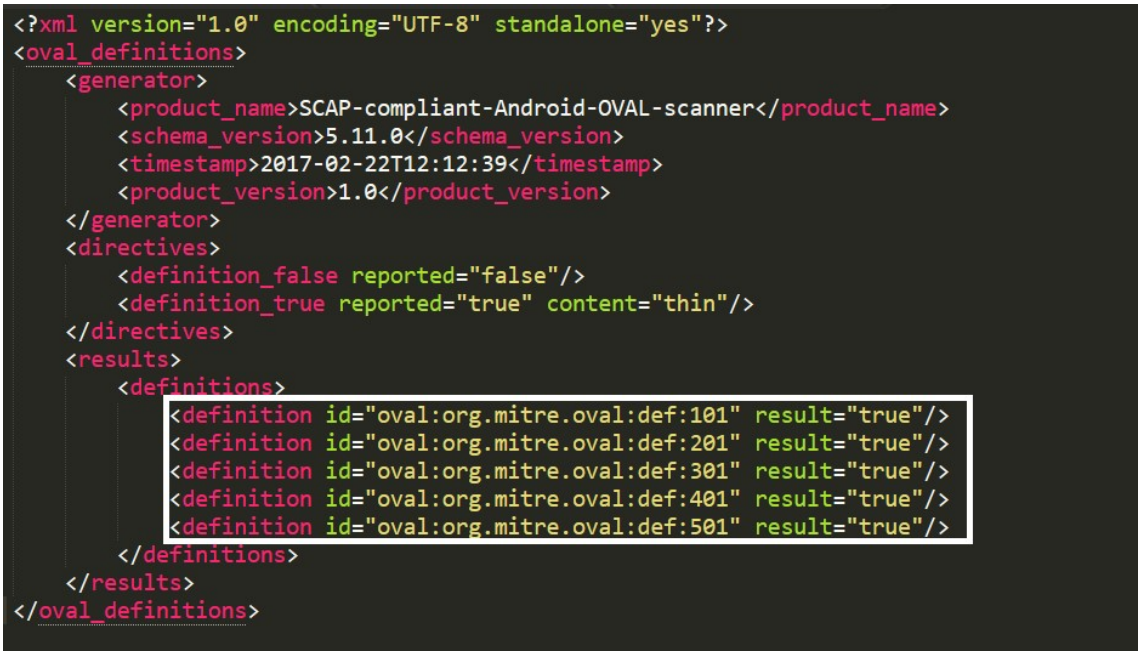


FIG. 5.6. Case 3: The result file

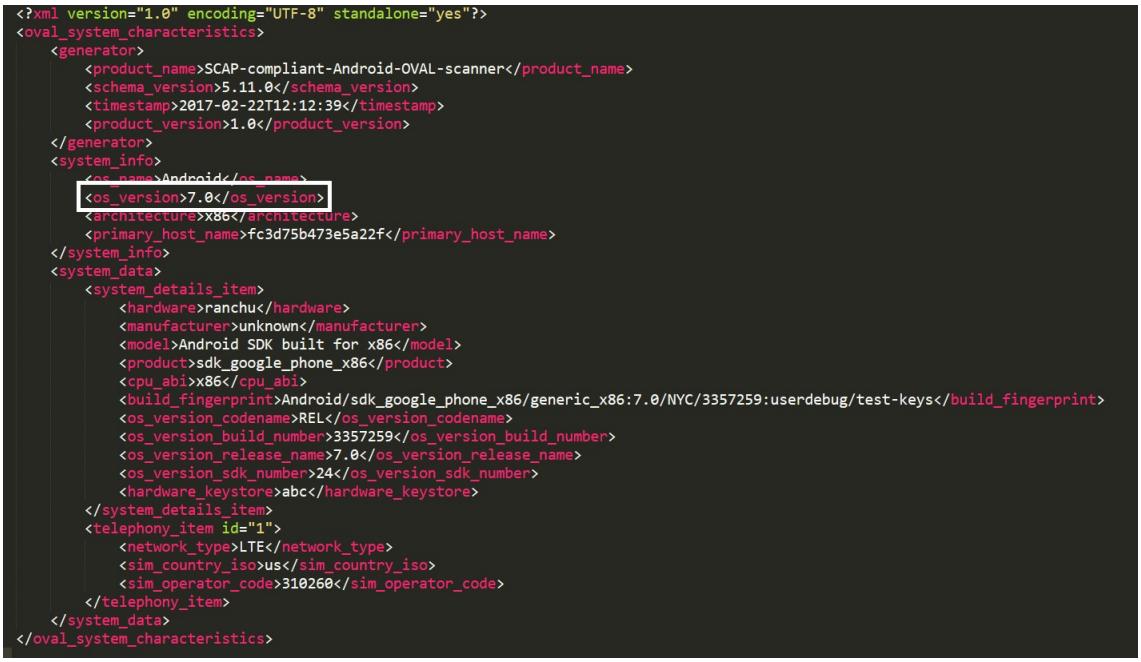


FIG. 5.7. Case 4: The system characteristics file

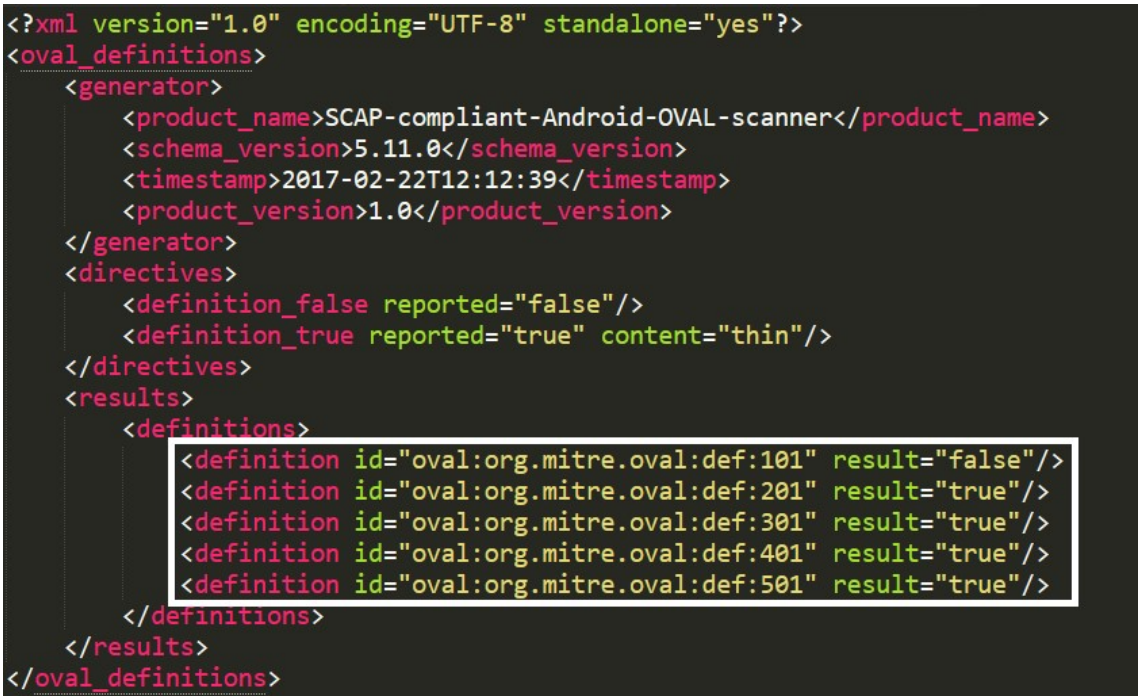
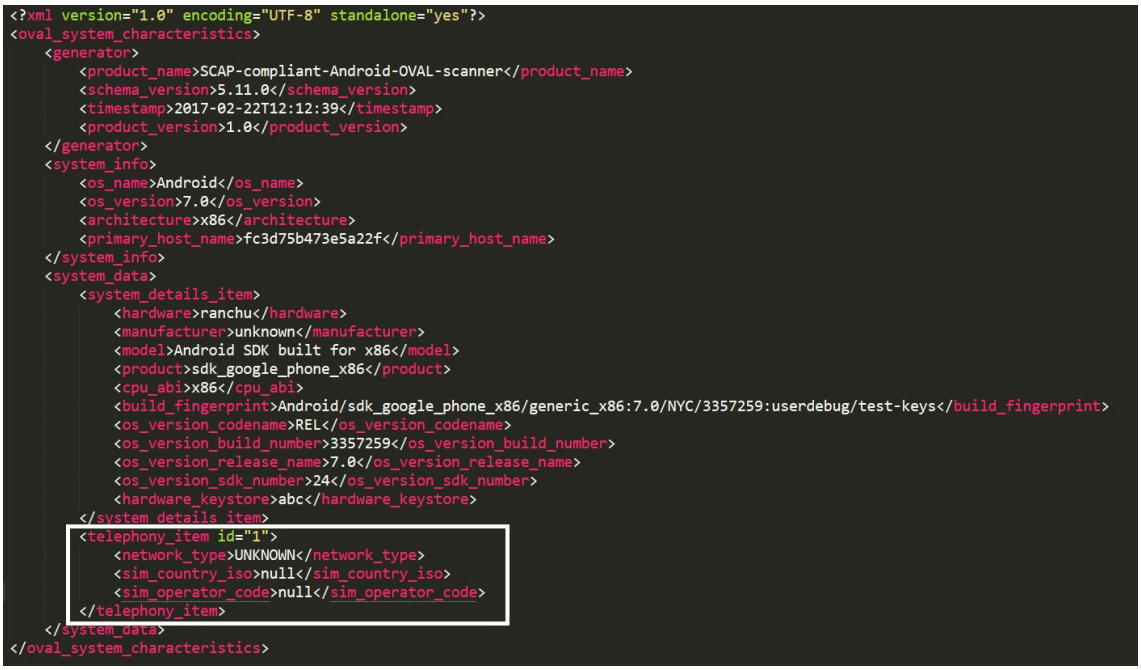
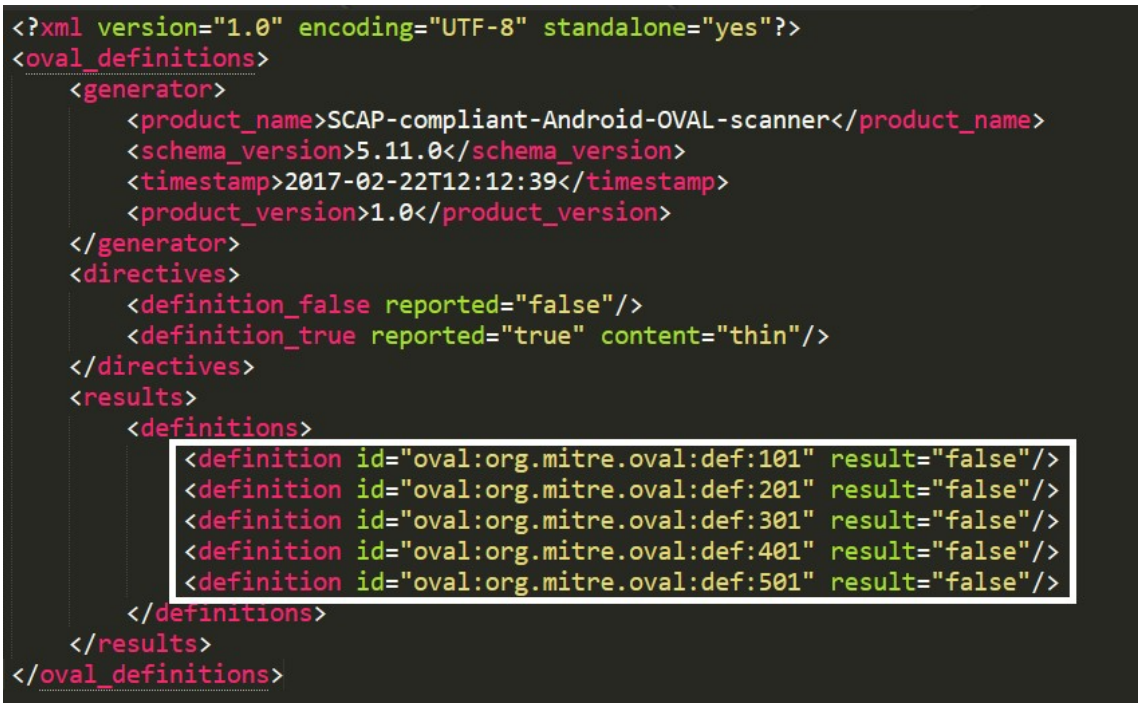


FIG. 5.8. Case 4: The result file



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<oval_system_characteristics>
    <generator>
        <product_name>SCAP-compliant-Android-OVAL-scanner</product_name>
        <schema_version>5.11.0</schema_version>
        <timestamp>2017-02-22T12:12:39</timestamp>
        <product_version>1.0</product_version>
    </generator>
    <system_info>
        <os_name>Android</os_name>
        <os_version>7.0</os_version>
        <architecture>x86</architecture>
        <primary_host_name>fc3d75b473e5a22f</primary_host_name>
    </system_info>
    <system_data>
        <system_details_item>
            <hardware>ranchu</hardware>
            <manufacturer>unknown</manufacturer>
            <model>Android SDK built for x86</model>
            <product>sdk_google_phone_x86</product>
            <cpu_abi>x86</cpu_abi>
            <build_fingerprint>Android/sdk_google_phone_x86/generic_x86:7.0/NYC/3357259:userdebug/test-keys</build_fingerprint>
            <os_version_codename>REL</os_version_codename>
            <os_version_build_number>3357259</os_version_build_number>
            <os_version_release_name>7.0</os_version_release_name>
            <os_version_sdk_number>24</os_version_sdk_number>
            <hardware_keystore>abc</hardware_keystore>
        </system_details_item>
        <telephony_item id="1">
            <network_type>UNKNOWN</network_type>
            <sim_country_iso>null</sim_country_iso>
            <sim_operator_code>null</sim_operator_code>
        </telephony_item>
    </system_data>
</oval_system_characteristics>
```


FIG. 5.9. Case 5: The system characteristics file



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<oval_definitions>
    <generator>
        <product_name>SCAP-compliant-Android-OVAL-scanner</product_name>
        <schema_version>5.11.0</schema_version>
        <timestamp>2017-02-22T12:12:39</timestamp>
        <product_version>1.0</product_version>
    </generator>
    <directives>
        <definition_false reported="false"/>
        <definition_true reported="true" content="thin"/>
    </directives>
    <results>
        <definitions>
            <definition id="oval:org.mitre.oval:def:101" result="false"/>
            <definition id="oval:org.mitre.oval:def:201" result="false"/>
            <definition id="oval:org.mitre.oval:def:301" result="false"/>
            <definition id="oval:org.mitre.oval:def:401" result="false"/>
            <definition id="oval:org.mitre.oval:def:501" result="false"/>
        </definitions>
    </results>
</oval_definitions>
```


FIG. 5.10. Case 5: The result file

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Thus,this research explored the possibility and the authenticity of building an SCAP compliant android vulnerability scanner for detecting the vulnerabilities reported because of the telephony feature , in the National Vulnerability Database (NVD). The android OVAL definition schema as provided by mitre.org was used to write the OVAL definition file which served as a guideline for the tests to be executed in order to determine whether the particular android operating system is vulnerable to these specific vulnerabilities. The scanning was carried out on a server to make the scanner apk lightweight by avoiding overhead caused by the bundling of jars like JAXB. The processing was thus carried out on the server , which involved the unmarshalling and marshalling of the XMLs. The system characteristics file which participates in the evaluation process was also generated and stored on the server , while the operating system details needed to encode this file were passed to the server as a parameter by the android device in consideration, while hitting request to the scanning service. The result of the vulnerability scanning, thus produced, was encoded back to results.xml file by the server and was stored on the server itself. The definition file, system characteristics file and the results file have been constructed in accordance with the OVAL schema for android as provided by the mitre.org. The scanner thus claims to be

SCAP compliant as it follows the OVAL standard which is a sub-standard of SCAP.

6.2 Future work

There is a lot of scope for future work in this domain as not much work has been done on making an SCAP compliant android vulnerability scanner. Since the scanner proposed , only detects telephony related vulnerabilities, it can be extended to include vulnerabilities because of other features in android like the Bluetooth, camera, mediaserver, etc. This would mean adding the appropriate definitions to the existing OVAL definition file proposed in this thesis and constructing the appropriate system characteristics file which will include system data for these features as well. The appropriate result file will be generated if these inclusions are made.

Another area in which which this thesis can take a direction is along with detection of vulnerabilities, carrying out the specified remediations as well. Several google bulletins have mentioned appropriate patches for many of the vulnerabilities, the details for which have been mentioned in the National Vulnerability Database (NVD) . If the scanner can be extended to detecting the vulnerabilities as well as applying the appropriate patches, then it would be a very useful feature to include.

REFERENCES

[1] Android vulnerabilities by Google. Google Android : List of security vulnerabilities.

[2] by mitre.org, S. OVAL Android Definition Schema Element Dictionary.

[3] Ere, M.; Hurel, G.; Badonnel, R.; and Festor, O. Increasing Android Security using a Lightweight OVAL-based Vulnerability Assessment Framework.

[4] Festor, O.; Ere, M.; Hurel, G.; and Badonnel, R. Increasing Android Security using a Lightweight OVAL-based Vulnerability Assessment Framework.

[5] Gurfinkel, D. 2016. Dirty Cow -Unauthorized Access.

[6] NIST. NVD - SCAP Validated Tools.

[7] Nvd.nist.gov. Agenda Process Model OVAL Results Tutorial The Basics.

[8] Oval.mitre.org. Writing an OVAL Definition.

[9] point research team, C. QUADROOTER.

[10] Schmidt, C. Technical Introduction to SCAP.

[11] Stagefright. Stagefright malware is back! 'Worst Android bug in history' returns for a third time — Daily Mail Online.

[12] Thomas, D. R.; Beresford, A. R.; and Rice, A. 2015. Security Metrics for the Android Ecosystem.

[13] Worrell, B. An Introduction to XCCDF.

[14] Xi-Salvador, L. 2008. A Survey on Languages, Enumerations and Other Tools used for Security Information Communication and Sharing.

[15] Zlabs. What is Quadrooter? Zimperium Mobile Security Blog.