# An Approach towards Standardising Vulnerability Categories

by

Yun Lillian Li

# An Approach towards Standardising Vulnerability Categories

by

**Yun Lillian Li**

**Dissertation**

submitted in fulfilment of the

requirements for the

**Master's Degree**

in the subject

**COMPUTER SCIENCE**

in the

**FACULTY OF ENGINEERING, BUILT ENVIRONMENT, AND INFORMATION TECHNOLOGY**

of

**UNIVERSITY OF PRETORIA**

**PROMOTORS: PROF. J.H.P. ELOFF**

**DR H.S. VENTER**

**MAY 2007**

## ABSTRACT

*Computer vulnerabilities are design flaws, implementation or configuration errors that provide a means of exploiting a system or network that would not be available otherwise. The recent growth in the number of vulnerability scanning (VS) tools and independent vulnerability databases points to an apparent need for further means to protect computer systems from compromise. It is important for these tools and databases to interpret, correlate and exchange a large amount of information about computer vulnerabilities in order to use them effectively. However, this goal is hard to achieve because the current VS products differ extensively both in the way that they can detect vulnerabilities and in the number of vulnerabilities that they can detect. Each tool or database represents, identifies and classifies vulnerabilities in its own way, thus making them difficult to study and compare. Although the list of Common Vulnerabilities and Exposures (CVE) provides a means of solving the disparity in vulnerability names used in the different VS products, it does not standardise vulnerability categories. This dissertation highlights the importance of having a standard vulnerability category set and outlines an approach towards achieving this goal by categorising the CVE repository using a data-clustering algorithm. Prototypes are presented to verify the concept of standardizing vulnerability categories and how this can be used as the basis for comparison of VS products and improving scan reports.*

# ACKNOWLEDGEMENTS

***I would like to express my sincere thanks to the following people for their assistance during the production of this dissertation:***

- ***Prof. J.H.P. Eloff and Dr H.S. Venter – thank you for your patience, inspiration and motivation.***
- ***Mr C. Naicker – thank you for all your support and advice.***
- ***The ICSA team (http://icsa.cs.up.ac.za/) – thank you for providing me an excellent research platform.***

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

# 1 INTRODUCTION

## *1.1 INTRODUCTION*

The Internet is a valuable part of our lives. It is used by millions of people every day from all across the globe to perform online transactions, search for useful information and communicate with other people. However, despite its power and value to us, its open and dynamic environment provides a perfect breeding ground for pernicious cyber-crimes and vicious security attacks. Five information security services are defined to implement security measures, namely Authentication, Confidentiality, Integrity, Availability and Non-Repudiation [GOLL 99]. Malicious attackers constantly look for and exploit weaknesses in computer systems in order to attack or break these security services. Those weaknesses in security systems that might be exploited to cause harm or loss are referred to as vulnerabilities [PFLE 03]. This makes Internet security a challenging but interesting topic to research.

In the growing world of computer and network security, constant progress has been made in tool development and techniques for both compromising and protecting computer systems. The increase in the number and quality of security products points to the apparent need for further ways to protect computer systems from compromise [BAKE 99]. Security experts have developed many information security technologies, for example Firewalls [TIWA 00], Intrusion Detection Systems [BACE 00], Vulnerability Scanners [HARN 02] and Cryptography [MCCL 02]. Each security technology implements one or more of the information security services mentioned above.

Vulnerability Scanners are popular proactive information security technologies. They search systems and networks for the occurrence of known flaws and produce a report containing all the search results that an individual or an enterprise can then use to strengthen its security. This process is called vulnerability scanning. Vulnerability scanners are proactive because they find vulnerabilities before an intruder has the opportunity to exploit them; thus they are preventative measures. They detect security flaws based on a database of known flaws. The vulnerability database is updated periodically as new vulnerabilities are discovered.

Vulnerability scanners are used widely in industry by many large corporates and organizations. The rapid growth rate of CVE shows evidence of the popularity of the technology. CVE was founded in 1999 and in early 2007, it already has 153 organizations participating, used by 273 products and services, and 73 organizations as Advisories [MITR 06]. However, several challenging problems have manifested in the field of this particular proactive security technology, some of which are discussed in the following section.

## 1.2 MOTIVATION FOR THIS RESEARCH

Despite the usefulness of vulnerability scanners, they currently pose a variety of problems. A major difficulty is that vulnerability scanners are disparate in the way that the vulnerabilities are named and organised in the vulnerability database of each respective vulnerability scanner. For example, one vulnerability scanner might call a particular vulnerability a “Trojan horse”, while another might call the same vulnerability a “virus”. Naming inconsistency can create confusion and misunderstanding to the users, and prevents collaboration between vulnerability scanners. The number of vulnerabilities in the vulnerability databases of vulnerability scanners also differs significantly. Furthermore, some vulnerability scanners define their own vulnerability categories. A vulnerability category refers to the grouping of specifically the same types of vulnerabilities. This complicates the problem even more. Some vulnerability scanners define a small set of vulnerability categories while others define many categories. For example, SFProtect defines seven vulnerability categories [SFPR 04]: User accounts, Audit policy, System logon, File system, Registry, Services, and Shares, whereas Nessus [NESS 06] defines more than 40 categories, which include: AIX Local Security Checks, Backdoors, Brute force attacks, CGI abuses, CISCO, Databases, and many more.

Lack of standards for vulnerability and vulnerability categories causes this disparity in the vulnerability database structures, which makes it very difficult to compare and assess vulnerability scanners. Another problem with current vulnerability scanners is that they create huge administrative burdens. After each scan has been completed, a detailed report is generated to list all vulnerabilities found and the recommendations for the corrective actions. Such reports are often large and place huge burdens on administrators to rectify the vulnerabilities. For example, the Nessus Security Scanner [NESS 06] provides a report that gives detailed and organised information about all the vulnerabilities detected on the network based on the address of the hosts, ports and security issues regarding the port. Such a report is poses numerous difficulties for an administrator as it is lengthy and does not highlight the problem areas of the network for a quick response. This could have a negative impact on the efficiency and effectiveness of risk management, as vulnerabilities are often not rectified immediately.

This research was, therefore, motivated by the problems referred to above. The next section of this chapter defines the problem statement of this research.

## *1.3 PROBLEM STATEMENT*

The research at hand recognises the importance of information security and specifically that of current vulnerability scanners. It is aimed principally at making a contribution towards enhancing vulnerability product assessment by having a standardised set of vulnerability categories to facilitate the decision-making process.

The problem can be addressed by considering the following research questions:

### What is the state of current vulnerability scanners (VS)?

Vulnerability scanners, which are also referred to as vulnerability assessment technologies, are proactive information security technologies and they attempt to search for known vulnerabilities before these can be exploited by intruders [VENT 03]. In order to improve on this technology, knowledge on its current status is required. This research therefore investigates the current status of vulnerability scanners, the different types of vulnerability scanners available, and their strengths and weaknesses.

### How can the disparity in current vulnerability scanners be solved?

Disparity is one of the major problems posed by current vulnerability scanners. They all name and categorise vulnerabilities differently, which makes VS assessment and comparison very difficult.

This research question therefore involves finding techniques to standardise the types of known vulnerabilities, so that it will be possible to know which subset of standardised vulnerability categories a specific vulnerability scanner can detect from a potentially exhaustive set of standardised vulnerabilities.

### How can vulnerability scanners be assessed effectively?

There is a disparity in current vulnerability scanners in the way that they detect vulnerabilities. For example, vulnerability scanner A is able to detect only a certain type of vulnerability, whereas vulnerability scanner B can detect different types of vulnerabilities. Due to such disparity, VS assessment and comparison cannot be carried out in a uniform way. Since there exists no real measurement for vulnerability product assessment, this research question involves finding a means that will facilitate such assessment. The assessment of vulnerability scanners allows for the best suitable tools to be selected for providing coverage rather than relying on a single vendor for a "suite" solution. It could also be possible to use an array of vulnerability scanners to protect the enterprise's network and assets where the weakness of one vulnerability scanner is the strength of another. This would, in turn, enhance the communication and security of organisations using vulnerability scanners.

### How can the administrative burden caused by current vulnerability scanners be improved?

Current vulnerability scanners scan for vulnerabilities and report on their findings after a scan has been completed. It is difficult and time consuming to attend effectively to the vulnerabilities reported after the vulnerability scan, because such reports are often very long and rectifying all the problems indicated would place an immense administrative burden on human resources.

The current research involves finding techniques to ease this administrative burden on human resources.

## *1.4 TERMINOLOGY USED IN THIS DISSERTATION*

It is important to interpret the terminology used in this dissertation correctly in order to avoid misunderstanding. Detailed terminology is defined when encountered throughout the dissertation. However, to ensure that the main terms are well defined, the researcher provides a brief definition of what is meant by the terms **information security, vulnerability, vulnerability scanning,** and **data clustering.**

### 1.4.1 Information security

Like all important business assets, information is an asset that has value in an organisation and consequently needs to be protected [BSIB 03]. Information security can be defined as measures adopted to prevent the unauthorised use, misuse, modification or denial of the use of assets [MAIW 03]. Furthermore, the objective of information security is not to protect assets, but it is the name given to the preventative measures that can be taken to safeguard assets [IFAC 98].

Five information security services are defined to implement information security measures. They are [GOLL 99]:

- **Authentication –** process of verifying a claimed identity
- **Confidentiality –** prevention of unauthorised disclosure of information
- **Integrity –** prevention of unauthorised modification of information
- **Availability –** prevention of unauthorised withholding of information
- **Non-repudiation –** concerned with providing proof of the origin such that the sender cannot deny sending a specific message, and the recipient cannot deny receiving that message

The unauthorised activities mentioned above consist of both deliberate and accidental threats.

### 1.4.2 Vulnerability and Vulnerability Scanning

Vulnerability is a state of being exposed to attack, injury, ridicule or litigation [SCHN 00]. In the context of this dissertation, a vulnerability is a design flaw or a configuration error that provides a means of exploiting a system or network that would not be available otherwise [HARN 02]. It could also be implementation error where a good design is realised poorly. Vulnerability can be either local or remote and it can attack on Confidenciality, Integrety, and Availability of a system. Details of vulnerability is discussed in a later chapter.

Vulnerability scanning is an information security technology implemented by a vulnerability scanner product to proactively identify vulnerabilities of computing systems in a network in order to determine if and where a system can be vulnerable [WEBO 04]. They are automated scanning programs that closely examine or scan computers and networks of computers to proactively detect known vulnerabilities [SCHN 00]. Vulnerability scanning is sometimes referred to as **vulnerability assessment** or **vulnerability management**. However, vulnerability management would include more aspects of scanning, such as patching and rectification of vulnerabilities. For this dissertation, the terms **vulnerability scanning** and **vulnerability scanner** are preferred and will be used throughout this dissertation.

### 1.4.3 Data Clustering

Clustering is a process of partitioning a set of data (or objects) in a set of meaningful sub-classes, called clusters [OSMA 05]. Data clustering, in the context of this dissertation, is a classification process that groups a set of data into clusters such that the intra-cluster similarity is maximised and the inter-cluster similarity is minimised.

The rest of the dissertation is outlined as discussed in the section that follows.

## *1.5 LAYOUT OF DISSERTATION*

This dissertation consists of eight chapters. C**hapter 1** provides an introduction to the research problem. Background of the research and motivations are also discussed in this chapter.

**Chapter 2** explains **vulnerability scanning** in detail, including its architecture. There are two kinds of vulnerability scanning: network-based and host-based vulnerability scanning. Each kind is explained in Chapter 2, together with its strengths and limitations. This chapter also investigates the current state of vulnerability scanners. The reader is provided with a list of popular vulnerability scanners that are currently available on the market.

**Chapter 3** focuses on vulnerability databases. Several popular vulnerability databases are discussed. The need for a common enumeration of vulnerabilities is then explained, and the Common Vulnerabilities and Exposures (CVE) list is introduced. This includes an explanation of the CVE list, why it is public, and the role of CVE. This chapter highlights the CVE since it attempts to solve the naming disparity in vulnerability scanners and enables the researcher to advance standardisation to the next level, which involves the standardisation of vulnerability categories.

**Chapter 4** provides an overview of data clustering and the researcher explains the motivation for using data clustering in this research. Different data-clustering techniques are discussed briefly and a typical data-clustering process is outlined in the chapter. One of the data-clustering techniques, namely the Self-Organising Feature Map (SOM), is discussed in greater detail since this particular technique has been selected for this research. A list of reasons is provided to support this decision.

In **Chapter 5**, the categorisation process is explained. Brief reference is made to related work done by various other researchers and institutions on vulnerability categorisation. The categorisation process proposed in this research involves constructing a standard set of vulnerability categories from the CVE list using the SOM and maintaining this set to be up to date. The process consists of two phases: Initial Categorisation and Classification. Initial Categorisation is a once-off initialisation for the categorisation process. The second phase, Classification, is dependent on the first phase. New vulnerabilities are found and published every day, and this phase shows how the categorisation process is capable of handling newly discovered vulnerabilities as they arise. The classification phase is an on-going process to maintain the categorised CVE database that has been created in the first phase.

**Chapter 6** explains the Vulnerability Analyser (VA) prototype. VA is developed by the researcher who implements the two-phased categorisation process as discussed in Chapter 5. The experimental results obtained from the VA are critically evaluated and the benefits of having a standardised set of vulnerability categories are discussed.

**Chapter 7** introduces the Vulnerability Product Assessor (VPA). Once the standard set of vulnerability categories have been constructed, then it is possible to develop a program that automates the process of vulnerability scanner assessment. In this chapter, a conceptual model for conducting automated vulnerability scanner assessment is proposed. The design of the model is discussed in detail and a prototype is presented to demonstrate the concept. Experimental results are also provided for the VPA.

**Chapter 8** summarises the research undertaken and explains the extent to which the research problem has been solved. The dissertation concludes with a reflection on possible areas for future research and improvements.

# CHAPTER 2

# 2 OVERVIEW OF VULNERABILITY SCANNING

## *2.1 INTRODUCTION*

The number of Internet users is increasing explosively, and the trend of more users accessing the Internet is expected to rise as time goes on. However, the number of illegal access and malicious hacker attackers is also increasing. Without a doubt, the costs of cyber-attacks are significant, as shown by the 2006 Computer Crime and Security Survey conducted by the Computer Security Institute and the FBI [FBIR 06]. The 639 organisations that participated in the eleventh annual study reported combined losses of $52.49 million, with causes ranging from theft of proprietary information, denial of service (DoS) attacks, viruses to unauthorised access. This indicates that computer security is a major problem. With the launch of the Internet, intranets and other open systems, security experts are becoming more aware of the necessity to assess and manage potential security risks in their environments. One of the big concerns for Information Security Managers is the management of vulnerabilities in the large networked Information System environment.

Vulnerabilities have been defined as design flaws, implementation or configuration errors that provide a means (which would otherwise not have been available) by which a system or network can be exploited [HARN 02]. Vulnerabilities can also be implementation errors where a good design is realised poorly. For example, a piece of software could be well-designed but development bugs are introduced when implementing the design which results in buffer overflows when being executed. Hackers use the Internet as their main medium for sharing information about exploiting software vulnerabilities. New vulnerabilities are discovered every day and the pace of discovery shows no signs of ever slowing down [ISTR 07]. Furthermore, attack tools are widely available, automating the exploit of one or several vulnerabilities with minimum effort and knowledge. Vulnerable machines can be exploited by hackers to gain administrative privileges, which can in turn be used to execute arbitrary codes or to launch DoS attacks against third-party systems. Vulnerable machines also enable malicious code to spread autonomously. The **confidentiality**, **integrity** and **availability** of the information assets are at risk if the information system infrastructure does not manage vulnerabilities properly. Managing vulnerabilities by deploying the right security safeguards will bring the organisation in a proactive security mode, rather than having it operate from a reactive mode where one has to constantly solve the problems as they are occurring. Vulnerability scanning technology plays a key role in this regard.

This chapter will address a proactive information security technology – **vulnerability scanning**. The next section describes vulnerability scanning in detail, including its architecture, the different types of vulnerability scanning, and the strengths and weaknesses of such technology. It also investigates the current state of vulnerability scanners. At the end of the section, a list of commercially available vulnerability scanners is provided, after which the chapter is concluded with some final remarks on vulnerability scanning.

## *2.2 VULNERABILITY SCANNING*

### 2.2.1 What is Vulnerability Scanning?

While public servers are important for communication and data transfer over the Internet, they open the door to potential security breaches by threat agents such as malicious hackers and intruders. Vulnerability scanners proactively search for security flaws based on a database of known flaws. They test systems for the occurrence of these flaws and generate a report of the findings that an individual or enterprise could use to tighten the network's security before hackers could exploit them. By stimulating controlled attacks that get stopped before they cause too much damage [ENTS 02], the scanners test each service that a computer offers to determine whether it is vulnerable to attack. In essence, they search for misconfigured application servers and network components (for example J2EE servers, Web servers, routers and switches) that are vulnerable to known flaws. They search for obsolete applications, and often for applications that are enabled by default but should be disabled, especially those with known problems (e.g. Remote-Procedural-call (RPC) services on UNIX or the User Datagram Protocol (UDP) ECHO program on Windows NT/2000).

A vulnerability scanning process typically involves the following three steps [SNYD 03]:

1. The first task in the scanning process is to find out which hosts are running on the network.
2. The second task is to find out which applications and services are running on the network and how are they configured.

3. Lastly, vulnerability scanners employ a lengthy sequence of tests to detect whether each separate system is vulnerable to a particular known bug or problem.

The three steps mentioned above may vary for different vulnerability scanners. Smarter vulnerability scanners such as Internet Scanner [ISSC 06] iterate between steps two and three to get more knowledge about the systems and use that to launch additional tests. Other vulnerability scanners like SAINT [SAIN 06] may decide to simplify the scanning to save scanning time and minimise the possibility of overloading the target systems [SNYD 03].

## 2.2.2 The Architecture of Vulnerability Scanners

There are not many types of vulnerability scanner architectures available in the market. Lopyrev [LOPY 01] defined an architecture for a distributed vulnerability scanner in which the network environment is divided into a list of disjoint network zones and each network zone has a scanning agent and a management console managed by the central management server and the central management console. For this research, Bace's [BACE 00] architecture for vulnerability scanners is most relevant, because it shows the importance of the vulnerability database to the vulnerability scanner. The architecture is shown in Figure 2.1.

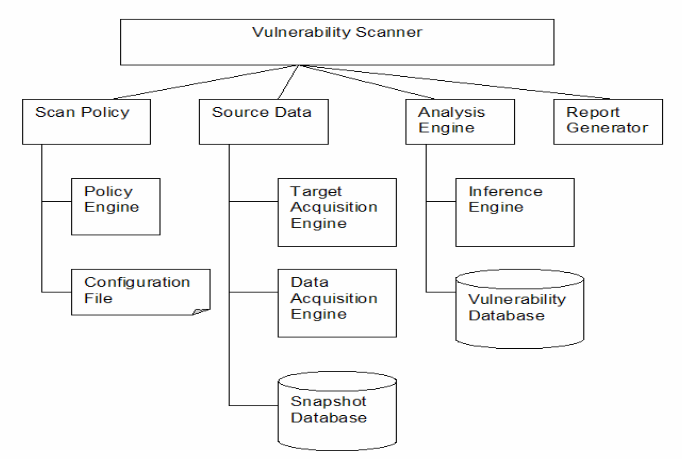


*Figure 2.1 A Typical Architecture for Vulnerability Scanner*

This architecture consists of four components, namely the **scan policy**, the **source data**, the **analysis engine** and the **report generator**.

The scan policy specifies how the vulnerability scanner is configured to scan for vulnerabilities within its network. For example, it may be configured to scan only certain hosts or certain specified types of vulnerabilities on its network to save network and system resources. User scan configurations are contained in the configuration file and the policy engine then manages the configuration file and loads it before every scan.

The source data serves as an input to the vulnerability scanner, which scans this input in a bid to find vulnerabilities. Vulnerability scanners can scan for vulnerabilities at two different levels: the host level and the application level. The different levels of data sources that such a scanner is able to scan are referred to as targets. Vulnerability scanners can scan two types of targets, namely host-based targets and application-based targets [COLE 02]. Host-based targets refer to individual hosts that a vulnerability scanner needs to scan and application-based targets are one or more specific applications running on a target host that need to be scanned by a vulnerability scanner. The source data contains three sub-components. The target acquisition engine searches for the specific target hosts to check whether a host is connected to the network or not. The data acquisition engine acquires systems' attributes and configurations, and stores them in a database (referred to as the Snapshot database).

The collected source data is used by the analysis engine to determine vulnerabilities. The analysis engine compares the source data with the vulnerability database, which contains signatures of all known computer vulnerabilities. The inference engine matches the Snapshot database with the vulnerability database in order to detect vulnerabilities. A vulnerability is found if a specific data string in the Snapshot database matches with the string in a vulnerability database.

The analysis engine detects all vulnerabilities and a detailed report about the findings is created by the report generator. Such reports can also contain additional information such as how and where to fix the vulnerabilities detected.

Now that the architecture of vulnerability scanners has been discussed, our attention in the next section turns to the two kinds of vulnerability scanning.

### 2.2.3 Network-based and Host-based Vulnerability Scanning

There are two kinds of vulnerability scanning: Network-based and Host-based vulnerability scanning. Each of them is explained in more detail below.

#### 2.2.3.1 Network-based Vulnerability Scanning

A network scanner is usually the first tool used in the vulnerability scanning process because it can detect extremely critical vulnerabilities such as a vulnerable web server in a Demilitarised Zone (DMZ) or misconfigured firewalls. The network scanner performs quick and detailed analyses – from an external or internal intruder's perspective – of the critical networks and system infrastructures of an enterprise [ISSS 04].

Network-based vulnerability scanning tools run on one computer and sends packets of data to other computers. When a response arrives, the scanner matches it with a predefined table of possible vulnerabilities, and based on that the result, determines whether a vulnerability exists. For example, to see if a vulnerable version of BIND exists on a computer, one can send a set of packets to the BIND server, make a query that contains the command version.bind, and BIND returns the version number. If it does not come back with the version number 8.2.2-p5, it proves that there is a vulnerability in the system [ENTS 02].

The strengths of network-based scanners can be summarised as follows:

- Detailed repair reports are generated speedily to facilitate prompt counteractive action when the analysis of the network-based devices is complete.
- They detect unknown or unauthorised devices and systems on a network, as well as unknown perimeter points on the network (e.g. unauthorised remote access servers or connections to insecure networks of business partners).
- The scanners provide a comprehensive view of all available operating systems and services running on the network, as well as a detailed listing of all system user accounts that standard network resources can find. This provides administrators with a clear picture of what types of services are indeed being used on their networks. This information can be used by network scanners for further vulnerability analysis.
- Network-based scanners are easy to set up and use because no host software needs to be installed on the systems being scanned. They test vulnerabilities of critical network devices that do not support host-scanning software such as printers, switches, routers, firewalls and remote access servers.
- They provide "real-world" scanning of systems that have already been locked down with host-based assessment tools such as databases, critical files, web and application servers and firewalls. Furthermore, they allow for the detection of critical configuration errors on these systems.

Network-based scanning products (e.g. routers, firewalls and web servers living outside the corporate firewall) can also evaluate the configurations and strengths of systems. These capabilities imply that network-based scanners should be used to scan entire networks.

The following subsection discusses the other kind of vulnerability scanning – host-based vulnerability scanning.

### 2.2.3.2 Host-based Vulnerability Scanning

While a network-based scanner imitates a network-based intruder, a host-based scanner tests a system from the perspective of a system user that has a local account on the system. This is the fundamental difference, since a network-based scanner, due to its nature, cannot provide sufficient understanding of potential user activity risks. User activity risks range from user ignorance to malicious misuse of the system. For example, an intruder could be an external hacker, or a legitimate user who has access to a particular host but misuses an account. Once the intruder has gained access to a local account, it has the opportunity to exploit and control the local system. Thus, managing user activity risks is critical to the security of the entire network, and not only to the affected host. Host-based scanners help to ensure that a particular host is configured properly and that its vulnerabilities are also patched so that a local user does not gain access to administrator or root privileges.

Host-based scanners can gain direct access to low-level details of specific services, configurations details and the operating system of a particular host. This property gives host-based scanners their strength in vulnerability scanning, which can be summarised as follows:

- Host-based scanners identify risky user activity that includes sharing directories with unauthorised parties; policy avoidance such as using weak passwords or no passwords; failure to run anti-viral software,  and bypassing the corporate firewall by using own dial-up connection.
- They detect signs of whether an intruder (internal or external) has already penetrated a system. These hacker trances include device files found in unexpected places, unexpected new files, files with suspicious file names, and unexpected programs that have potentially obtained “root” privilege.
- Host-based scanners can create hashing (e.g. MD5) baselines for critical files. This allows administrators to compare the current files on a system to a formerly known secure state.

- They detect whether an intruder is still active on a system by locating “sniffer” programs or other unauthorised services that are currently running on the system (e.g. file transfer servers and Internet Relay Chat (IRC)).
- Host-based scanners detect well-known hacker back-door programs and they can also detect local host services vulnerable to “local buffer overflow” exploits from hacker web sites. Without a host-based scanning process in place, malicious low-level users can easily download these exploits from popular hacker web sites (e.g. www.rootshell.com), compile and then run them to obtain access to full “root” level privileges.
- They perform security tests (such as password guessing and policy checks) that are unfeasible or difficult for a network scanner to perform. They can check network services, including Network File System (NFS), Hyper Text Transfer Protocol Daemon (HTTPD), and FTP, to ensure they are being correctly configured and implemented. Host-based scanners are best suited for performing resource-intensive file system and baseline scans, which are impractical with network-based scanners because entire contents of hard drives would need to be transferred to the scanning system over the network.
- Host-based scanners can detect more security flaws for Windows hosts than network-based scanners, due to the great number of available “back-door” and “sniffer” programs for these operating systems.

Host-based scanners are an obvious choice for securing systems that host critical files, directories, databases and other critical application services. These systems often contain critical business data, and host-based scanning can detect high-risk vulnerabilities and offer correction information to make sure that local system users do not have inappropriate access to these services and resources. Host-based scanning should also be a priority for securing firewall products. These products usually run on standard NT or UNIX operating systems and they often contain well-known vulnerabilities and default configurations as they ship from the vendor.

Now that both network-based and host-based vulnerability scanning have been explained in detail, the next section will discuss the strengths and weaknesses of vulnerability scanning tools in general.

### 2.2.4 The Current State of Vulnerability Scanners

The current state of vulnerability scanners is discussed in terms of the known strengths and weaknesses of this technology, which are summarised in the two subsections that follow.

### 2.2.4.1 Current strengths of vulnerability scanners

Vulnerability scanners are valuable because they identify vulnerabilities in the selected components of infrastructure, thereby enabling the security manager to better evaluate risk and develop adequate security protections.

The benefits of using vulnerability scanning tools are summarised below [ENTS 02]:

- Vulnerability scanners are equivalent to potential intruders because they look for the same vulnerabilities that hackers identify and exploit. Vulnerability scanners identify known security flaws before potential intruders do.
- They answer the **What, Where,** and **How** of the network's security vulnerabilities. They furthermore discover what the vulnerabilities are on the network, where the vulnerabilities are located and how they can be fixed. Penetration test procedures enumerate live network services, identify remote operating systems, and determine existing holes and patch levels.
- Vulnerability scanners aid in producing or updating a comprehensive network map of the enterprise. An enterprise should have a precise picture of what systems are present in its environment. Machines that are unofficially connected to the network can introduce unnecessary additional risks into the enterprise and need to be dealt with promptly.
- The scanners assist in creating a register of all the devices on an enterprise's network. The register could contain information such as device types, hardware configurations, operating system levels, application versions, vulnerabilities associated with the devices and applications, and any other relevant system information.
- They provide knowledge of those particular systems in the enterprise's perimeter that are vulnerable to the infection process of a specific worm and sketch the overall security bearing of the enterprise. Knowledge of risk exposure is essential so as to make the right decisions within a limited time frame.
- Vulnerability scanners assist administrators to stay up-to-date with security risks and countermeasures. If any level of automation is provided by the vulnerability scanner, it will reduce the administrative burden on its administrator for staying abreast of new security vulnerabilities – provided that the vendor supplies timely and reliable updates.

### 2.2.4.2 Current weaknesses of vulnerability scanners

Despite their usefulness, vulnerability scanners are plagued by many problems that are summarised below [ENTS 02; HARN 02; VENT 03]:

- **False negatives**: False negatives are known vulnerabilities that exist on the target system, but they are not reported by the vulnerability scanner products. False negatives incorrectly give administrators a sense of security even though vulnerabilities do exist.
- **False positives:** False positives are vulnerabilities that are reported by the scanner but that actually do not exist on the target system. False positives waste administrators' time on manually analysing and correcting non-existent vulnerabilities.
- When scanning a large network, vulnerability scanners frequently find so many problems that correcting them becomes an overwhelming job. The scan report can be very lengthy and this is often the reason why there are still so many vulnerable systems hosting well-known critical problems. It is simply because there are too many vulnerabilities and administrators do not have enough time to fix them all.
- From a usability point of view, current vulnerability scanners have limitations in terms of "easy installation", "easy operation" and "intelligible scanned results". Almost all security scanners for Windows are easy to install because they have their own installers, but almost all for UNIX and Linux require compiling, for example the Nessus Scanner. Many scanners are command line based; some require compiling and other need complex command line options. The scanned results and security-related information are not easily understood by new users. As a result of poor usability, vulnerability scanners are not intuitive and require a steep learning curve for new users.
- Many services are perfectly secure when offered in isolation, but when combined with other services result in an exploitable vulnerability. For example, if file transfer protocol (FTP) and hypertext transfer protocol (HTTP) services are hosted on the same machine, an attacker can use the ftp service to write data to a directory that the web server can read, and it may be possible for the attacker to cause the web server to execute a program written by the attacker. Currently, many vulnerability scanners do not attempt to discover how combinations of configurations on the same host or between hosts on the same network can contribute to the vulnerabilities of the network.
- Vulnerability scanners themselves have vulnerabilities. Many require root access when scanning. A server being scanned can be configured to act in response to a scanner in a way to potentially exploit vulnerabilities on the scanner machine.
- Vulnerability scanners consume bandwidth. Although typical vulnerability scanners do not generate more than 10 Mb/s of network traffic, this usage can choke a slower network such as a dial-up or ISDN link.
- They may crash devices on the network. When a scan starts, the scanner tries to detect the hosts that are connected to the network and the services they are running. This typically

involves trying every possible combination of target host address and the potential services they offer. As a result, a network device will now find a large amount of traffic to hosts it may not be familiar with and also on ports it may not know about. This additional complexity might unintentionally exploit code or architecture flaws of the network device that were not previously encountered in its development or test cycles. The consequence is device failures.

- Servers that are on the network during scanning can be crashed by vulnerability scanners. A common bug reported when performing vulnerability scans is that a system does not crash when it gets scanned for vulnerabilities, but seems to lose its network stack and cannot communicate any more. FreeBSD [FREE 06] is an example of such an operating system; it has this sort of behaviour as a feature.
- Applications that are not robust enough can be crashed during scanning. For example, a Linux server may have excellent stability during a scan, but one test may cause a crash of its Apache web server. Certain tests for buffer overflows and other vulnerabilities may unintentionally create DoS conditions in other applications.
- As discussed already, vulnerability scanners are disparate in the way that the vulnerabilities are named and classified in the vulnerability database of each different scanner. This is due to lack of standards for vulnerability and vulnerability categories. Such naming and classification disparity makes it very difficult when selecting vulnerability scanners for an organisation.

### 2.2.5 Commercially Available Vulnerability Scanners

A non-exhaustive list of vulnerability scanners that are commercially available either as freeware or commercial software is listed below:

- Saint [SAIN 06] is a commercial network-based vulnerability scanner that runs only on Linux/Unix platform. Saint scans at a host-level and anything with an IP address running TCP/IP protocols. It has dynamic reporting capabilities that allow drilling down on a specific vulnerability to obtain more information and how to correct the vulnerability. Saint can scan both IPv4 and IPv6 addresses, however, it is not very good at detecting any TCP/IP network services on non-standard ports (ports other than 80 or 8080). Saint is CVE-compatible (CVE is discussed in the next chapter).
- Nessus [NESS 06] is a free network-based vulnerability scanner. It is a full-featured and comprehensive vulnerability scanner that offers many configuration options, and it does an excellent job of identifying vulnerabilities. However, it is less robust in reporting and

providing guidance on correcting problems than are other commercial scanners such as Internet Scanner. Nessus is high performance and has a very low false positive rate. It is also good at finding TCP/IP network services running on non-standard ports.

- Internet Scanner [ISSC 06] is a commercial network-based vulnerability scanner that scans both at host and at application level. It performs scheduled and selective probes of the communication services, operating systems, key applications and routers of an organisation network, in search of those vulnerabilities most often used by unscrupulous threats to probe, investigate and attack the network. Internet Scanner then analyses the vulnerability conditions and provides a series of corrective actions, trends analyses, configuration reports and data sets. Internet Scanner is particularly good at detecting Windows vulnerabilities. It has a fairly low occurrence of false negatives but its performance is slower. It is also CVE-compatible.
- QualysGuard [QUAL 06] is a commercial, CVE-compatible, network-based, host-level vulnerability scanner. It is ideal for large, distributed networks that have an unlimited number of IP addresses, appliances and users. QualysGuard is delivered as a service over the web, thus it eliminates the burden of maintaining vulnerability management software. Clients securely access QualysGuard through an easy-to-use Web interface.
- Retina [RETI 06] is a commercial network-based vulnerability scanner that scans only at host level. It is designed to scan any machine on the Internet, intranet, or extranet network in order to identify existing vulnerabilities and check adherence to established security policies. It is a versatile, well-organised and full-featured vulnerability scanner offering vulnerability scans for Linux, Unix and Windows platforms, as well as automatic fixes of many detected problems. It also has the ability to create audits.

The above mentioned vulnerability scanners all suffer from the weaknesses outlined in section 2.2.4.2 to a greater or lesser degree.

## *2.3 CONCLUSION*

This chapter provided an overview of vulnerability scanning and gave a detailed discussion of its strengths and weaknesses. It introduced the two types of vulnerability scanning available, namely network-based scanning and host-based scanning. Network-based scanning discovers and provides rectification information for network-based vulnerabilities. It secures network perimeters and reinforces first lines of defence against attackers. Host-based scanning provides a supplementary level of security by locking down hosts on the network to prevent critical resources from being accessed by internal misuse, or external intruders using compromised user accounts. A scan report is generated after the scan to report on the findings and it may further contain the corrective actions for those vulnerabilities.

Chapter 2 answered the first research problem: what is the state of current vulnerability scanners? The current state of vulnerability scanners, the different types of vulnerability scanners available, and their strengths and weaknesses were discussed. However, according to Bruce Schneier, “Security is not a product, it’s a process” [SCHN 99]. A vulnerability scanner alone merely provides a list of the vulnerabilities that it is able to identify on a system; if the administrators do not act upon what it reports, then the security state will not change. Adequate resources and processes are required to analyse the vulnerability reports, evaluate the risk, identify adequate solutions and apply necessary corrective actions. Vulnerability scanners need to regularly update their vulnerability database with newly discovered vulnerabilities. Regular assessment is essential in the ever-evolving infrastructure to maintain the expected security state.

As discussed before, vulnerability databases are the key component in vulnerability scanners because they contain all the known vulnerabilities that the vulnerability scanners can detect. However, they are disparate in the way that they name and organise their vulnerabilities. More details on this disparity will be discussed in the next chapter. It causes many problems for the current vulnerability scanners and one of the key drivers for this research is precisely to address this disparity. In the next chapter, vulnerability databases and the disparities among them are discussed in detail.

# CHAPTER 3

# 3 OVERVIEW OF VULNERABILITY DATABASES AND THE COMMON VULNERABILITIES AND EXPOSURES (CVE) LIST

## *3.1 INTRODUCTION*

In the growing world of computer and network security, development of tools and techniques are continuously improved to both compromise and protect computer systems. The increase in the number and quality of security products indicates the need for further ways to protect computer systems from compromise [BAKE 99].

Many vulnerability scanners contain their own vulnerability databases, and others use vulnerability databases from well-accredited external sources. A vulnerability database is a comprehensive database of vulnerabilities that contains information on vulnerabilities. It may also contain additional information such as links to patch information or corrective actions for the vulnerabilities. Vulnerability information is critical for the protection of information systems. For example, enterprises need to know whether components of their existing or planned computing environment are vulnerable to security failure, and software developers need early warnings when their products have security flaws. A standardised and comprehensive vulnerability list is valuable to security experts for them to build products and services accordingly, and system administrators want information on relevant security flaws and their resolutions. The quality of the vulnerability scanners is directly dependent on the quality of its vulnerability database. Furthermore, by exchanging, interpreting and correlating information about vulnerabilities among various scanners, collaboration can be achieved. It is obviously very hard to achieve this. One of the major obstacles to achieving collaboration is the lack of a common enumeration of computer vulnerabilities. Many vulnerability scanners adopt their own naming scheme for their vulnerabilities. As a result, it causes disparity among the scanners, which makes it very difficult to compare and assess them. As referred to earlier, one vulnerability scanner might call a particular vulnerability a “Trojan horse”, while another might call the same vulnerability a “virus”.

This chapter reviews three popular public vulnerability databases – US-CERT, Open-Source Vulnerability Database (OSVDB) and Bugtraq. At present, these databases are the most dominant in the market. Many vulnerability scanners have references to these vulnerability databases, for example Retina [RETI 06] has references to the Bugtraq database. These databases also reference one another. This cross-reference approach among vulnerability databases would quickly become cumbersome, because it requires order of $n^2$ mappings where $n$ represents the number of vulnerability databases. This creates a need for a common enumeration of vulnerabilities, and the CVE List is therefore introduced. An explanation is given of what the CVE list is, why it is public and the role of CVE. Chapter 3 concludes by explaining the motivation for choosing the CVE list as the data source for clustering.

## 3.2 THE US-CERT NOTES DATABASE

The Department of Homeland Security and various public and private sectors jointly form the United States Computer Emergency Readiness Team (US-CERT) [CERT 06]. It was established in September 2003 with the aim to improve computer security awareness and respond to cyber-attacks in the United States. US-CERT is the national central point for preventing, protecting against, and responding to cyber-security incidents and vulnerabilities. It interacts on a continuous basis with all state and local governments, federal agencies, the research community, private industry, and others to broadcast cyber-security information.

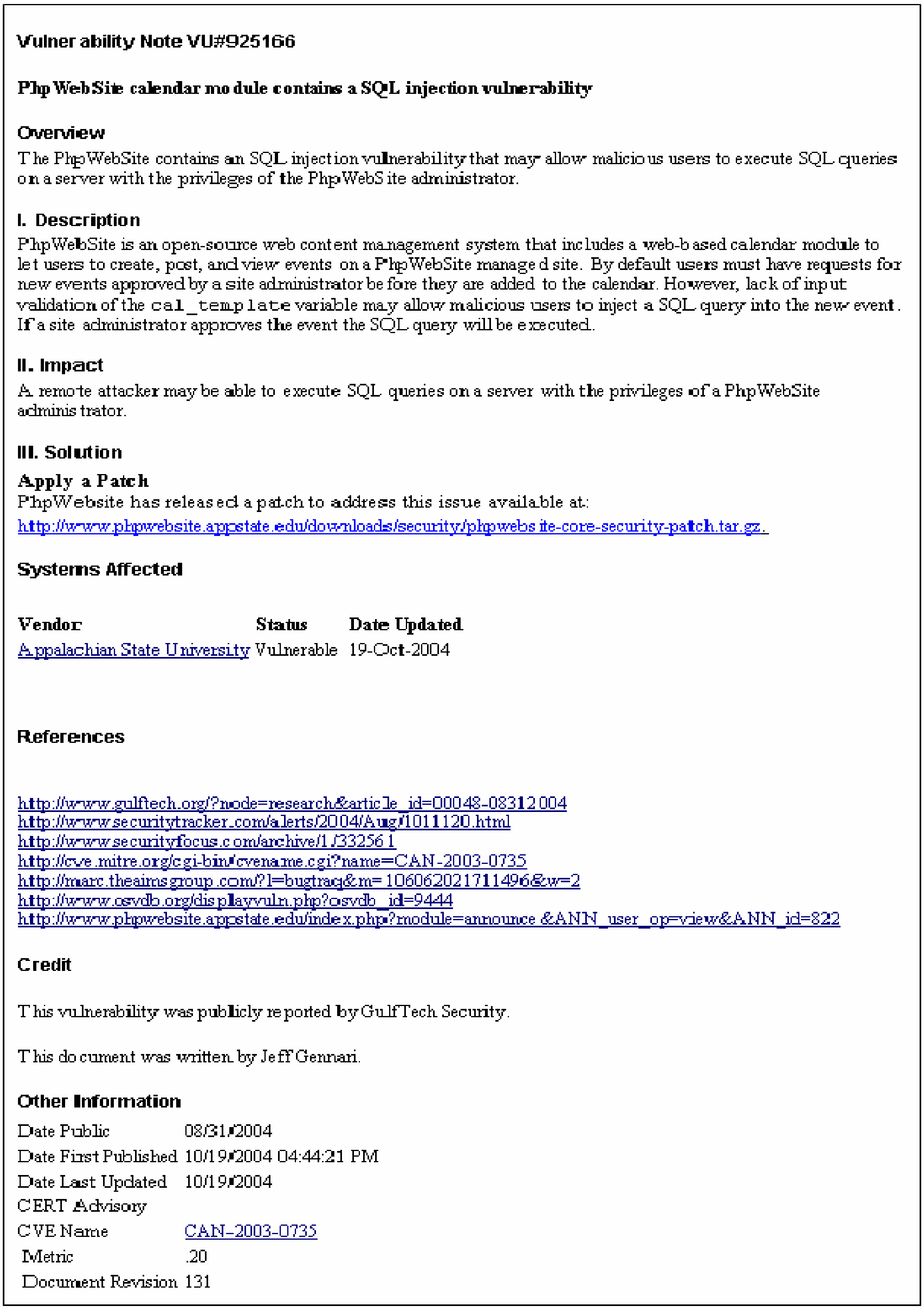
US-CERT publishes information on a wide variety of vulnerabilities. The US-CERT Notes database contains two types of documents: Vendor Information documents, which provide information about a specific vendor's resolution to a particular vulnerability, and Vulnerability Notes, which describe vulnerabilities in general and independent of a particular vendor [CERT 06].

The Vendor Information document provides the following information: the date that US-CERT notified the vendor of the vulnerability or the earliest date on which the vendor is known to have become aware of the vulnerability; the last updated date of the vulnerability; a status summary to indicate whether the vendor has any products that US-CERT considers to be vulnerable; the vendor's official response to US-CERT queries about the vulnerability, and US-CERT's comments on the vulnerability. For example, Microsoft is such a vendor with a range of products that have vulnerabilities, such as Internet Explorer, Windows Media Player, Windows XP, Windows Server 2003, etc. [CERT 06].

The Vulnerability Notes document contains the following information about the vulnerability [CERT 06]: an assigned vulnerability ID; a vulnerability name; a revision number; an abstract that provides a summary of the vulnerability and its impact; a description; an impact statement; a solution statement; a list of vendor systems that may be affected; a collection of URLs to external references; the people who have initially discovered the vulnerability; the date(s) when the vulnerability first became known; when US-CERT first published the Vulnerability Notes; when the vulnerability note was last updated, and a metric number that indicates the severity of the vulnerability. Figure 3.1 shows an example of a US-CERT Vulnerability Notes document [CTVU 06].

By July 2007, US-CERT contains 2289 vulnerabilities in its database. Although the US-CERT Vulnerability Notes database is updated constantly with new vulnerabilities, it covers only a subset of vulnerabilities. US-CERT tracks neither Trojan horse programs nor viruses in its database. Although, Trojan horses and viruses can be types of attacks that may exploit vulnerabilities rather than being vulnerabilities in their own right, they open back doors and further weaken the security of a system. Thus, in this research, they are regarded as vulnerabilities because their existence poses a threat to the security of an organization. Furthermore, US-CERT does not classify its vulnerabilities into different vulnerability categories. The US-CERT Vulnerability Notes database is cross-referenced with the CVE catalogue. The CVE Name field in the Vulnerability Notes document allows the user to easily cross-reference vulnerabilities described by US-CERT with those from the CVE list. The CVE list and its importance are discussed in more detail in the latter section of this chapter.

The next section discusses another public vulnerability database – the Open-Source Vulnerability Database – in more detail.



**Vulnerability Note VU#925166**

**PhpWebSite calendar module contains a SQL injection vulnerability**

**Overview**

The PhpWebSite contains an SQL injection vulnerability that may allow malicious users to execute SQL queries on a server with the privileges of the PhpWebSite administrator.

**I. Description**

PhpWebSite is an open-source web content management system that includes a web-based calendar module to let users to create, post, and view events on a PhpWebSite managed site. By default users must have requests for new events approved by a site administrator before they are added to the calendar. However, lack of input validation of the `cal_template` variable may allow malicious users to inject a SQL query into the new event. If a site administrator approves the event the SQL query will be executed.

**II. Impact**

A remote attacker may be able to execute SQL queries on a server with the privileges of a PhpWebSite administrator.

**III. Solution**

**Apply a Patch**

PhpWebsite has released a patch to address this issue available at:
http://www.phpwebsite.appstate.edu/downloads/security/phpwebsite-core-security-patch.tar.gz.

**Systems Affected**

| Vendor | Status | Date Updated |
|---|---|---|
| Appalachian State University | Vulnerable | 19-Oct-2004 |

**References**

http://www.gulftech.org/?node=research&article_id=00048-08312004
http://www.securitytracker.com/alerts/2004/Aug/1011120.html
http://www.securityfocus.com/archive/1/332561
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0735
http://marc.theaimsgroup.com/?l=bugtraq&m=106062021711496&w=2
http://www.osvdb.org/displayvuln.php?osvdb_id=9444
http://www.phpwebsite.appstate.edu/index.php?module=announce&ANN_user_op=view&ANN_id=822

**Credit**

This vulnerability was publicly reported by GulfTech Security.

This document was written by Jeff Gennari.

**Other Information**

| | |
|---|---|
| Date Public | 08/31/2004 |
| Date First Published | 10/19/2004 04:44:21 PM |
| Date Last Updated | 10/19/2004 |
| CERT Advisory | |
| CVE Name | CAN-2003-0735 |
| Metric | .20 |
| Document Revision | 131 |

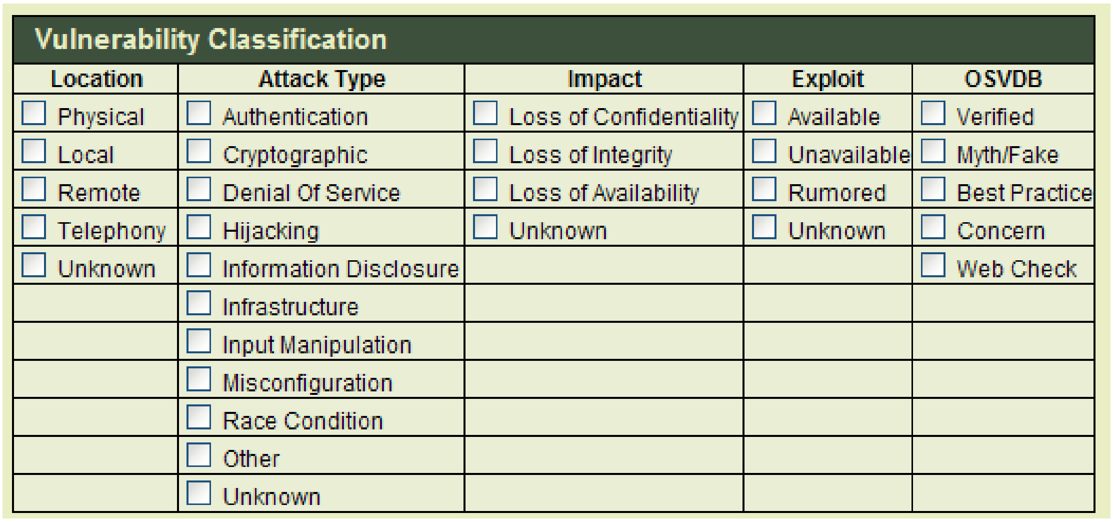
*Figure 3.1 An example of a US-CERT Vulnerability Notes Document*

## 3.3 THE OPEN-SOURCE VULNERABILITY DATABASE (OSVDB)

The Open-Source Vulnerability Database (OSVDB) project was launched in 2002 after the security community realised that there is no independent, community-operated vulnerability database existing on the market. On 31 March 2004, the OSVDB opened for public use [OSVD 06]. The project has gained significant acceptance and recognition since then, and for this reason, the OSVDB is included in this discussion of vulnerability databases.

The OSVDB maintains an independent database of computer security vulnerabilities created by and for security professionals and projects freely around the world [OSVD 06]. The goal is to provide detailed, accurate, up-to-date and unbiased technical information about all types of vulnerabilities. The OSVDB is currently a web application. It has a front end part that allows users to search or report vulnerabilities, and a back-end part that allows contributors to add or edit vulnerabilities. OSVDB promotes better, more open collaboration among organisations and individuals, reduces redundant work, and decreases expenses inherent in the development and maintenance of proprietary vulnerability databases. It is completely community-based, and although this is its biggest strength, it has other strengths too. It is unbiased and neutral in its process of discovering, accepting, verifying and publishing vulnerabilities. Its openness in terms of acceptance of community input and internal reviewing processes ensures that the OSVDB is not biased by any vendor. The OSVDB team strives to ensure that content published reflects the true representation of vulnerabilities that neither over-expose nor under-expose any specific operating systems, products or vendors.

The OSVDB organises and verifies vulnerability data so that users can easily use them. It also includes both business-level and technical-level descriptions for the vulnerabilities in the database. The OSVDB creates and supplies the proper type of information for all consumers of vulnerability information. Its features and services benefit the whole security community because the OSVDB is universally available, without any distribution controls or fees. Although it references other vulnerability databases and the CVE list, it constructs its own database records to ensure that vulnerability data can be distributed and re-used without any restrictions. As of March 2006, OSVDB contains more than 11 000 vulnerabilities in its database, ranging across all categories [OSVD 06]. Vulnerabilities are not classified into a single vulnerability category; instead, the OSVDB has several classification criteria. It classifies vulnerabilities according to locations, attack types, impact, exploit information, and some OSVDB-specific classifications that are not found in most other sources. Figure 3.2 shows the complete set of the OSVDB’s vulnerability classification [OSCT 06].



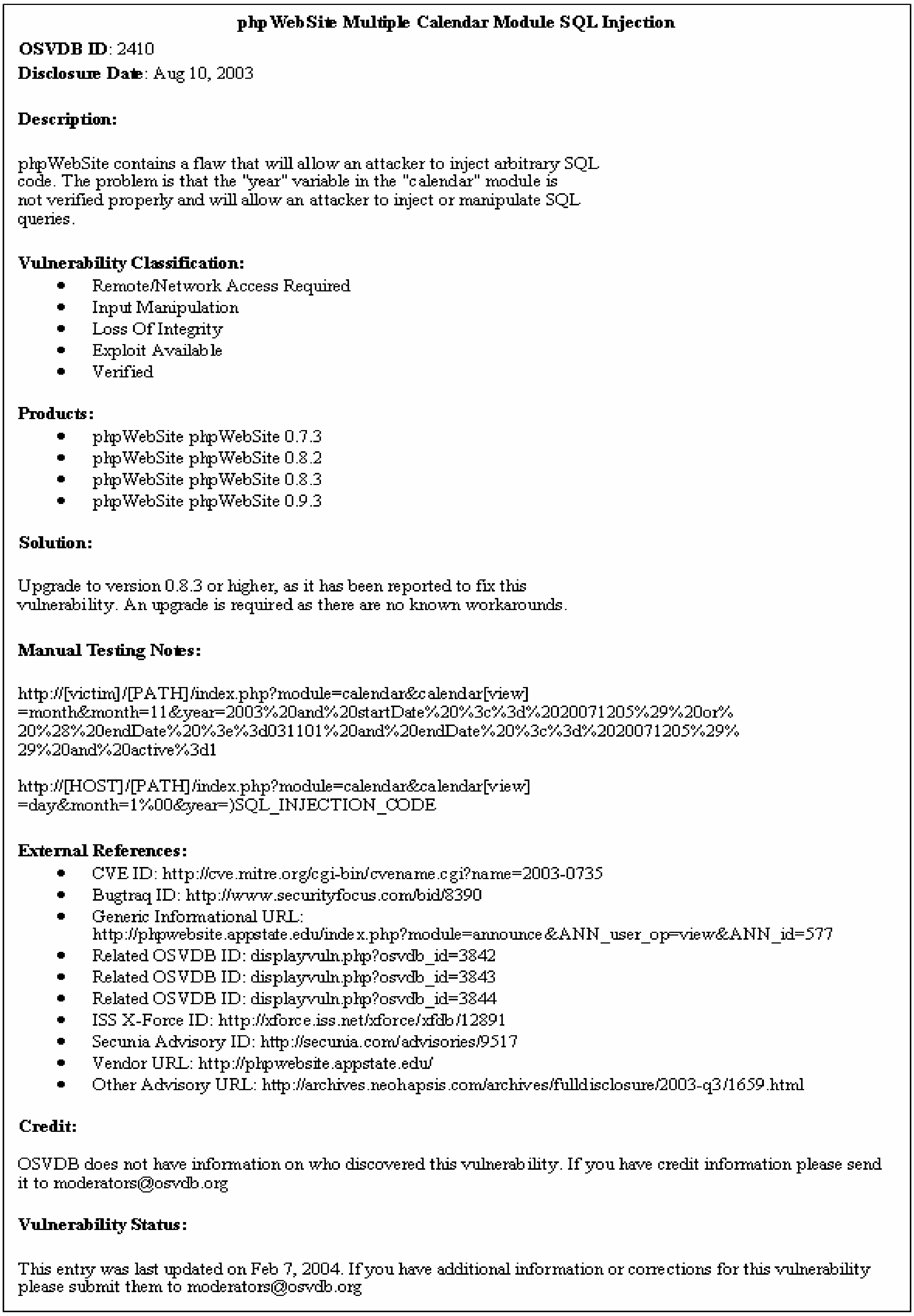
Vulnerability Classification

| Location | Attack Type | Impact | Exploit | OSVDB |
|---|---|---|---|---|
| ☐ Physical | ☐ Authentication | ☐ Loss of Confidentiality | ☐ Available | ☐ Verified |
| ☐ Local | ☐ Cryptographic | ☐ Loss of Integrity | ☐ Unavailable | ☐ Myth/Fake |
| ☐ Remote | ☐ Denial Of Service | ☐ Loss of Availability | ☐ Rumored | ☐ Best Practice |
| ☐ Telephony | ☐ Hijacking | ☐ Unknown | ☐ Unknown | ☐ Concern |
| ☐ Unknown | ☐ Information Disclosure | | | ☐ Web Check |
| | ☐ Infrastructure | | | |
| | ☐ Input Manipulation | | | |
| | ☐ Misconfiguration | | | |
| | ☐ Race Condition | | | |
| | ☐ Other | | | |
| | ☐ Unknown | | | |

***Figure 3.2 OSVDB Vulnerability Classification***

The fields of OSVDB records are very similar to those of the US-CERT Vulnerability Notes document. Figure 3.3 is an example of an OSVDB record of the same vulnerability illustrated in Figure 3.1 for US-CERT Vulnerability Notes [OSID 06]. OSVDB records are derived from US-CERT notes, and as a result OSVDB vulnerability names and descriptions are similar to those in US-CERT. OSVDB nevertheless contains a vulnerability classification section that is not in US-CERT Vulnerability Notes.

The next section discusses the Bugtraq vulnerability database, which is maintained by SecurityFocus [SECU 05].



**phpWebSite Multiple Calendar Module SQL Injection**

**OSVDB ID**: 2410

**Disclosure Date**: Aug 10, 2003

**Description:**

phpWebSite contains a flaw that will allow an attacker to inject arbitrary SQL code. The problem is that the "year" variable in the "calendar" module is not verified properly and will allow an attacker to inject or manipulate SQL queries.

**Vulnerability Classification:**

- Remote/Network Access Required
- Input Manipulation
- Loss Of Integrity
- Exploit Available
- Verified

**Products:**

- phpWebSite phpWebSite 0.7.3
- phpWebSite phpWebSite 0.8.2
- phpWebSite phpWebSite 0.8.3
- phpWebSite phpWebSite 0.9.3

**Solution:**

Upgrade to version 0.8.3 or higher, as it has been reported to fix this vulnerability. An upgrade is required as there are no known workarounds.

**Manual Testing Notes:**

```
http://[victim]/[PATH]/index.php?module=calendar&calendar[view]
=month&month=11&year=2003%20and%20startDate%20%3c%3d%2020071205%29%20or%
20%28%20endDate%20%3e%3d031101%20and%20endDate%20%3c%3d%2020071205%29%
29%20and%20active%3d1

http://[HOST]/[PATH]/index.php?module=calendar&calendar[view]
=day&month=1%00&year=)SQL_INJECTION_CODE
```

**External References:**

- CVE ID: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2003-0735
- Bugtraq ID: http://www.securityfocus.com/bid/8390
- Generic Informational URL: http://phpwebsite.appstate.edu/index.php?module=announce&ANN_user_op=view&ANN_id=577
- Related OSVDB ID: displayvuln.php?osvdb_id=3842
- Related OSVDB ID: displayvuln.php?osvdb_id=3843
- Related OSVDB ID: displayvuln.php?osvdb_id=3844
- ISS X-Force ID: http://xforce.iss.net/xforce/xfdb/12891
- Secunia Advisory ID: http://secunia.com/advisories/9517
- Vendor URL: http://phpwebsite.appstate.edu/
- Other Advisory URL: http://archives.neohapsis.com/archives/fulldisclosure/2003-q3/1659.html

**Credit:**

OSVDB does not have information on who discovered this vulnerability. If you have credit information please send it to moderators@osvdb.org

**Vulnerability Status:**

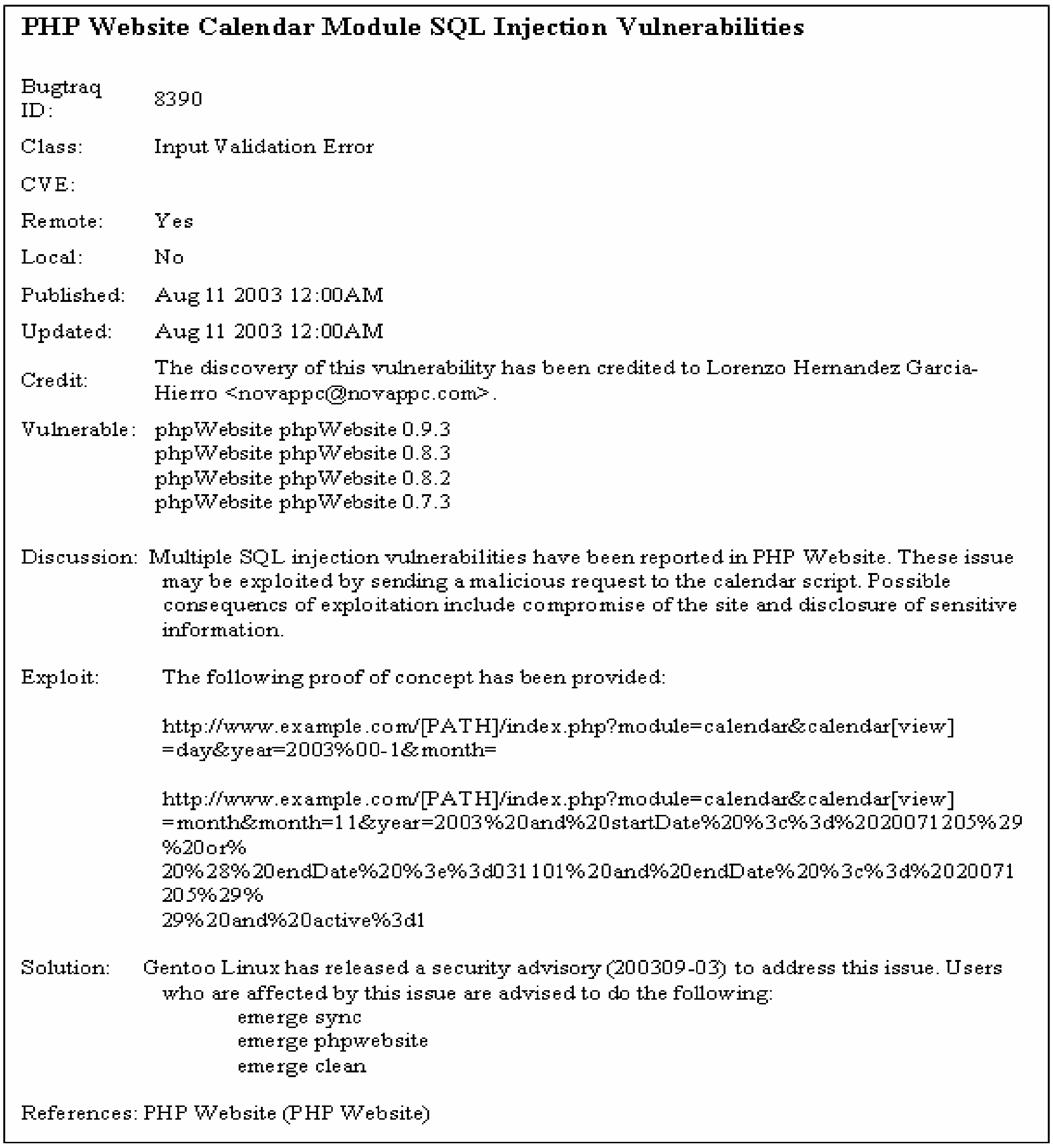
This entry was last updated on Feb 7, 2004. If you have additional information or corrections for this vulnerability please submit them to moderators@osvdb.org

*Figure 3.3 An example of an OSVDB vulnerability record*

## 3.4 THE BUGTRAQ DATABASE

The Bugtraq vulnerability database includes more than 23 000 entries on software-related vulnerabilities [BUGT 07]. It is hosted by SecurityFocus [SECU 05], and breaks down the vulnerabilities into 12 defined vulnerability categories. These are Serialisation Errors; Race Condition Errors; Origin Validation Errors; Input Validation Errors; Failure to Handle Exceptional Conditions; Environment Errors; Design Errors; Configuration Errors; Boundary Condition Errors; Atomicity Errors; Access Validation Errors, and Unknown Errors [BUGT 04]. Among these categories, there are five dominant categories: Input Validation Errors (23%); Boundary Condition Errors (21%); Design Errors (18%); Failure to handle exceptional conditions (11%), and Access Validation Errors (10%). These categories are dominant because they include the most prevalent vulnerabilities such as buffer overflows (included under boundary condition errors), and format string vulnerabilities (included under input validation errors). Similar to OSVDB, a vulnerability in Bugtraq database can also be classified into one or more categories.

Each vulnerability in the Bugtraq database has a unique Bugtraq ID number. Because the Bugtraq mailing list is the source of origin for most public vulnerability information, SecurityFocus [SECU 05] is currently negotiating with various vendors to start linking their tools back to the Bugtraq ID. Each vulnerability report in the database contains five sections: the info section; the discussion section; the exploit section; the solution section, and the credit section. The info section provides basic information of the vulnerability, such as BugTraq ID, vulnerability class, CVE Name, date of discovery, indication of whether this vulnerability is remotely or locally exploitable, version numbers of the vulnerable software, and version numbers of the software that are not vulnerability. The discussion section provides information for the cause of vulnerability and the consequences of successful exploitation of this vulnerability. The exploit section provides information on possible exploits. The solution section provides some possible solution to this vulnerability and the credit section provides information on who discovered the vulnerability as well as the external references for it. Figure 3.4 is an example of a BugTraq vulnerability for the same vulnerability illustrated in Figure 3.1 for US-CERT vulnerability note [BUGT 06].



**PHP Website Calendar Module SQL Injection Vulnerabilities**

| | |
|---|---|
| Bugtraq ID: | 8390 |
| Class: | Input Validation Error |
| CVE: | |
| Remote: | Yes |
| Local: | No |
| Published: | Aug 11 2003 12:00AM |
| Updated: | Aug 11 2003 12:00AM |
| Credit: | The discovery of this vulnerability has been credited to Lorenzo Hernandez Garcia-Hierro <novappc@novappc.com>. |
| Vulnerable: | phpWebsite phpWebsite 0.9.3<br>phpWebsite phpWebsite 0.8.3<br>phpWebsite phpWebsite 0.8.2<br>phpWebsite phpWebsite 0.7.3 |

Discussion: Multiple SQL injection vulnerabilities have been reported in PHP Website. These issue may be exploited by sending a malicious request to the calendar script. Possible consequencs of exploitation include compromise of the site and disclosure of sensitive information.

Exploit: The following proof of concept has been provided:

```
http://www.example.com/[PATH]/index.php?module=calendar&calendar[view]
=day&year=2003%00-1&month=

http://www.example.com/[PATH]/index.php?module=calendar&calendar[view]
=month&month=11&year=2003%20and%20startDate%20%3c%3d%2020071205%29
%20or%
20%28%20endDate%20%3e%3d031101%20and%20endDate%20%3c%3d%2020071
205%29%
29%20and%20active%3d1
```

Solution: Gentoo Linux has released a security advisory (200309-03) to address this issue. Users who are affected by this issue are advised to do the following:

```
emerge sync
emerge phpwebsite
emerge clean
```

References: PHP Website (PHP Website)

*Figure 3.4 An example of a BugTraq vulnerability record*

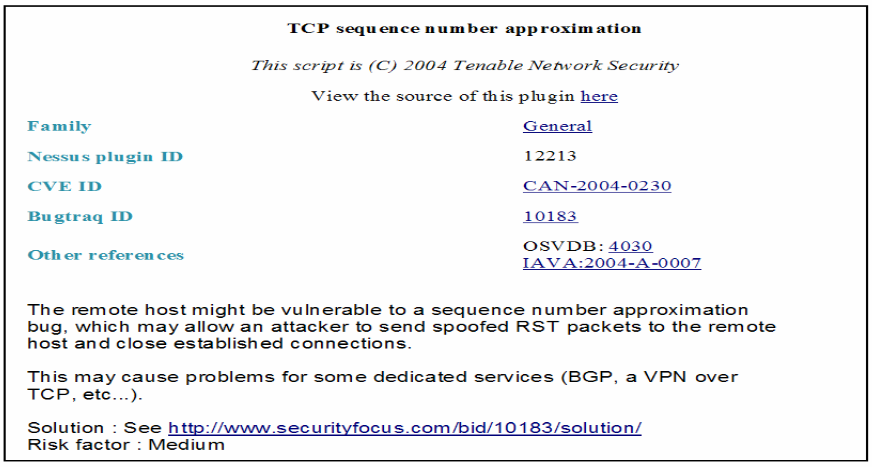
All three of the above vulnerability databases are public databases that are not specific to any vulnerability tools. In the next section, the Nessus Scanner vulnerability database will be discussed in detail, which will further illustrate just how extensive the difference among vulnerability databases can be.

## 3.5 NESSUS PLUGINS

Nessus is a popular open-source remote vulnerability scanner that is powerful, up-to-date, easy to use and free [NESS 06] .Vulnerabilities are called plugins in the Nessus scanner and they are simple programs that check for a given flaw. Nessus updates plugins daily. Plugins may include various external references such as US-CERT, BugTraq and vendor-specific databases.

Each plugin's information can be viewed on the Nessus website [NESS 06]. This includes plugin name, plugin family, an assigned Nessus plugin ID, CVE ID, BugTraq ID and other references, a brief description, possible solution, and risk factor for this plugin. A link to the source script of the plugin is also provided.

Nessus does not have the particular vulnerability as provided in Figure 3.1 for US-CERT vulnerability note. Figure 3.5 therefore shows the information of a Nessus plugin as it is displayed on the Nessus website [NESP 06]:



**TCP sequence number approximation**

*This script is (C) 2004 Tenable Network Security*

View the source of this plugin here

| | |
|---|---|
| Family | General |
| Nessus plugin ID | 12213 |
| CVE ID | CAN-2004-0230 |
| Bugtraq ID | 10183 |
| Other references | OSVDB: 4030<br>IAVA:2004-A-0007 |

The remote host might be vulnerable to a sequence number approximation bug, which may allow an attacker to send spoofed RST packets to the remote host and close established connections.

This may cause problems for some dedicated services (BGP, a VPN over TCP, etc...).

Solution : See http://www.securityfocus.com/bid/10183/solution/
Risk factor : Medium

***Figure 3.5 An example of a Nessus plugin***

As of March 2006, Nessus contains over 10 000 plugins in its database, covering 4 693 unique CVE entries and 4 510 unique BugTraq entries [NESS 06]. Nessus classifies its plugins according plugin families. A total of 45 plugin families have so far been defined and plugin families can be added or removed whenever there is a need to do so [NESS 06].

This section briefly discussed US-CERT Vulnerability Notes, OSVDB, BugTraq and Nessus Plugins. All these vulnerability databases are well recognised and frequently updated with new vulnerabilities. However, disparity already lies among them. Certain databases cover only a subset of vulnerabilities. For example, US-CERT tracks no viruses or Trojan horse programs in its database, while BugTraq contains only software-related vulnerabilities. Users of this database would have to consult other databases in case they need this information. Also, Nessus does not contain the particular vulnerability illustrated in Figure 3.1 at all in its plugin. In addition, each database gives different names to its vulnerabilities. For example, Nessus names the CVE entry "CAN-2004-0230" as "TCP sequence number approximation" (Nessus Plugin ID 12213), while US-CERT names it "The Border Gateway Protocol relies on persistent TCP sessions without specifying authentication requirements" (VU#415294). Besides the CVE list, there is no other standard source of reference for these databases to correlate, which makes the CVE list important for information sharing.

The disparity among various vulnerability databases has created a demand for a common enumeration of vulnerabilities, which is the basic reason for the emergence of CVE. The next section explains the CVE in more detail.

## 3.6 THE COMMON VULNERABILITIES AND EXPOSURES (CVE) LIST

There are many vulnerability scanners on the market. In order to protect a computer system, users need to be aware of what vulnerabilities various scanners can and cannot detect, and what vulnerabilities exist in the system. Those vulnerabilities need to be eliminated immediately and the risks of the remaining vulnerabilities need to be understood and managed.

In order to communicate this information to users, many vulnerability scanners include a database of security vulnerabilities. As discussed in previous sections, there exists significant disparity among these databases, and because they do not share a common element such as a vulnerability name, it is almost impossible to determine when different scanners are indeed referring to the same vulnerability. The consequences are as follows:

- Evaluating vulnerability scanners in a detailed fashion with respect to individual vulnerabilities will be difficult.
- Security experts face a steep learning curve every time they begin to use a new vulnerability scanner.
- Correlation among vulnerability scanners is difficult to achieve.
- Relating defensive information published by other sources to the vulnerabilities in question can be a difficult task.

- Productive discussions on defensive actions for vulnerabilities such as countermeasures are difficult to achieve.

Interoperability can be achieved by developing individual mappings among various vulnerability scanning products. This approach would quickly become cumbersome, because it requires order of $n^2$ mappings where $n$ represents the number of vulnerability databases (see Figure 3.6 [MANN 99]). Another approach to achieve interoperability is to create a standardised list of vulnerabilities to use as a mediator and perform only $n$ database mappings, as shown in Figure 3.7 [MANN 99]. The MITRE Corporation took the initiative to design and implement such an intermediary, known as the CVE list. The CVE list contains a reference list of common vulnerability and exposure names, which can be used or referenced in other tools and databases. The basic idea and approach of the CVE initiative was presented in January 1999 at the 2nd Workshop on Research with Security Vulnerability Databases at Purdue University [MANN 99].

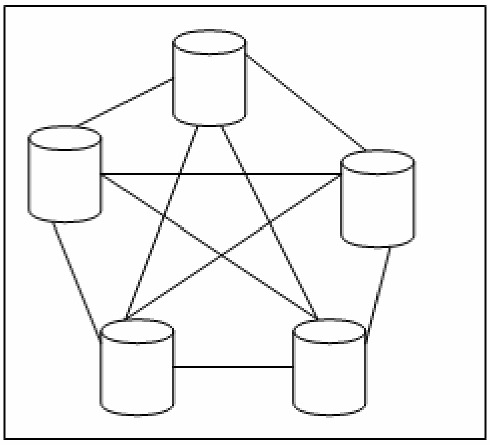


***Figure 3.6 Mapping each database to every other database***

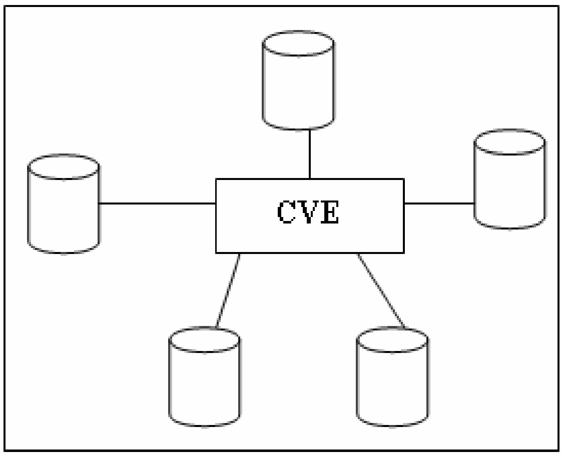


***Figure 3.7 Mapping each database to CVE***

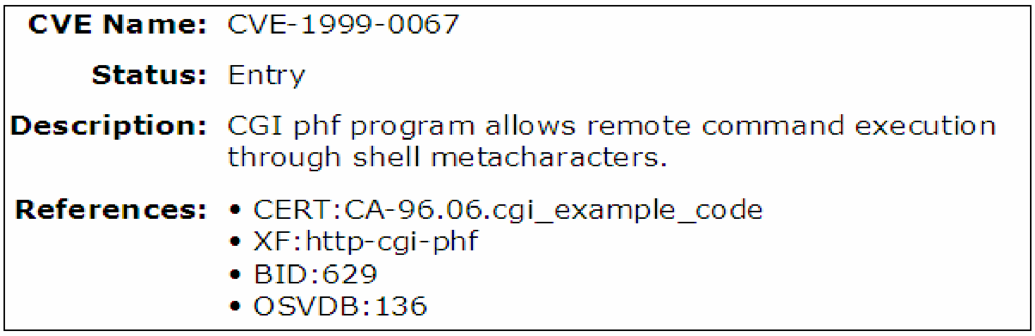
MITRE envisioned CVE to be an intermediary for collaborating vulnerability-related databases or concepts. In order to avoid competing with existing and future commercial efforts, CVE decided to limit its role to a logical bridge between other sources.

CVE was initiated in 1999 [MITR 06]. It pursues the adoption of a common naming convention for describing computer vulnerabilities and for referencing these names within security tools, advice web sites and security services, as well as commercial and open-source software package providers. It defines a standardised list that [MITR 06] aims to do the following:

- Differentiate and enumerate among all vulnerabilities that are publicly known.
- Identify each vulnerability with a standard, unique name.
- Exist independently of other resources that define vulnerability.
- Be publicly open and sharable without distribution restrictions.

With approximately 150 organisations participating with the CVE and more than 260 security products and services either CVE compatible or intending to be CVE compatible, the CVE has rapidly become an intermediary organising mechanism that makes the labour-intensive task of managing information security vulnerabilities more of an engineered practice [MITR 06].

On 19 October 2005, CVE had a one-time-only modification to the CVE list number scheme where the previous CVE list and the Candidate list are combined and a status attribute is added to the CVE entries to indicate whether they are official CVE entries or just candidates [MITR 06]. For this research, the focus is on the official CVE entries, thus the old CVE format is used. As illustrated in Figure 3.8, CVE uses a simple representation to record vulnerabilities: a CVE name, a status, a short description and a reference. A CVE name is a key assigned to a particular vulnerability, thus it must be unique. The status designates whether that CVE entry has Candidate, Entry or Deprecated status. Candidate status means that the CVE entry is under review by the CVE Editorial Board. Entry status indicates that it is an official CVE entry, and Deprecated status means that the CVE entry is no longer supported by the CVE Editorial Board. The description is concise and readable so that a human reader can have enough information to distinguish between vulnerabilities. The reference indicates the pertinent references for the CVE entry. References are other publicly available documents as stipulated by external organisations, which describe the vulnerabilities. Users can use the reference to gather more information on a particular vulnerability if the information given by CVE is not sufficient. The CVE name is an encoding of the year and a unique number *N*, where *N* represents the $N^{th}$ name assigned in that year with a prefix of “CVE". For example, CVE-1999-0002 is listed in 1999 and is the 2nd name assigned in that year. The key does not represent the order in which vulnerabilities are discovered or the severity of the vulnerability; it is only the chronological order in which the name was assigned in CVE.



| | |
|---|---|
| **CVE Name:** | CVE-1999-0067 |
| **Status:** | Entry |
| **Description:** | CGI phf program allows remote command execution through shell metacharacters. |
| **References:** | • CERT:CA-96.06.cgi_example_code<br>• XF:http-cgi-phf<br>• BID:629<br>• OSVDB:136 |

***Figure 3.8 An example of a CVE entry***

Although the description is concise, it does not contain detailed information that is usually associated with a particular vulnerability in other databases such as background or overviews, and it does not intend to provide any educational value for end users such as system administrators. This property is useful when it comes to run data clustering, which will be discussed in more detail later in this dissertation.

CVE has evolved into a cooperative, cross-industry, international, 35-organisation CVE Editorial Board-led effort to create and maintain the standardised CVE list of vulnerabilities and exposures.

There are five key features that underlie the CVE [MART 03]:

- Each vulnerability or exposure has one and only one name and standardised description.
- The CVE list acts as a dictionary for vulnerabilities rather than a vulnerability database.
- The CVE enables disparate databases and tools to “speak” the same language.
- The CVE list and other relevant information about CVE activities are publicly accessible via the web for downloading and reviewing.
- Industry endorsement occurs via a CVE Editorial Board and the development of CVE-compatible products and services.

CVE is the key to information sharing for vulnerability scanners. The common name simplifies the job of sharing data across separate databases and tools that were not easily integrated. If a scan report from one vulnerability scanner incorporates CVE names, then one can quickly and accurately access more information in various CVE-compatible databases to remedy the problem.

The CVE and its content structure have been explained in detail in this section, and the following subsection will now discuss why CVE has to be publicly available for everyone.

### 3.6.1 Why is CVE public?

The objectives of the CVE list are to provide a comprehensive enumeration for all publicly known vulnerabilities by assigning to each of them a unique and standard name. CVE can only achieve interoperability and communication if the list is openly available to the public, without any restrictions on distribution. Extensive commercially available vulnerability databases exist in the market but most of them are protected from open use by copyrights.

Any public discussion of vulnerability information may undoubtedly help intruders to compromise a system. However, the following are the reasons why the benefits of a publicly available CVE overshadow its risks [MANN 99]:

- CVE is limited to containing only publicly known vulnerabilities.
- Information sharing is more intricate within the security community than it is for intruders. For example, commercial databases have copyright restrictions precluding their use.
- It requires much more effort for an enterprise to protect its computing environment and eliminate all possible security flaws than for an intruder to discover and exploit a vulnerability to compromise the system.
- Community opinion is moving towards information sharing. This is shown by the fact that the CVE Editorial Board consists of key organisations in the security community.

Tool interoperability can be enhanced by having a widely accepted common enumeration for vulnerabilities. It will also enable more effective communication and create a more correlated security posture, thereby reducing the overall risk of security compromises.

The next subsection discusses the CVE Editorial Board as well as the roles of Board members.

### 3.6.2 CVE Editorial Board

CVE is industry-endorsed via the CVE Editorial Board. The CVE Editorial Board is responsible for the CVE naming process, and decides on which vulnerabilities to add to the CVE (discussed later in the chapter). The Board includes members from numerous information security-related organisations including commercial security tool vendors, members of academia, research institutions, government agencies and other prominent security experts. Through open and collaborative discussions, the Board identifies which vulnerabilities or exposures are to be included in CVE and then decides on a common name and description for each entry.

All Board members are expected to perform certain tasks pertaining to consultation and awareness. Consultation includes participation in board meetings and/or discussions of CVE content or Editorial Board processes. Awareness entails participation in board meetings, reading meeting summaries, and/or frequent reading of posts on the Editorial Board mailing lists.

There are four different roles for the Editorial Board members. Some board members may have more than one role, but all members must have only one primary role. The roles are [EDRO 04]:

- Technical
- Liaison
- Advocate
- Emeritus

Technical members participated in the design, development, review, application and maintenance of CVE. Liaison represents an important constituency, related to or affected by CVE, in areas that might not have technical representations on the Board. A liaison may even represent an organisation such as a particular software vendor. Advocates are leaders in the security community who actively support or promote CVE. They help bring credibility to CVE and give CVE a wider spectrum outside of the security community. Formerly active and influential members in the CVE initiative are known as Emeritus members. As a result of significant contributions, they maintain an honorary position on the Board. They may participate periodically in any technical, liaison or advocate tasks.

The MITRE Corporation formed the Editorial Board; it also moderates board discussions and ensures that CVE serves the public interest [MITR 06]. The minutes of Board meetings and history of discussions are available for review on the CVE website. Other security experts might be invited to participate on the Board as needed, based upon recommendations from Board members.

The CVE Editorial Board has been discussed in this section; the next subsection explains the process of adding new entries to the CVE list.

### 3.6.3 The CVE Naming Process

As stated earlier, CVE entries are defined by the CVE Editorial Board through open and collaborative discussions. The Editorial Board decides which vulnerabilities or exposures will be included in the CVE list and then determines the key, description and references for each entry.

Figure 3.9 graphically depicts the CVE Naming Process. The first step in this process is the identification of a potential vulnerability or exposure. This potential vulnerability is then assigned a CVE number, given the status of Candidate by the CVE, and proposed to the Editorial Board by a CVE Editor. CVE candidates are those vulnerabilities or exposures under consideration for acceptance into CVE.

There are two ways in which new security issues can become candidates [MITR 06]. The prime way is through Candidate Numbering Authorities (CNAs). CNAs are organisations that distribute CVE candidate numbers to researchers and information technology vendors for inclusion in first-time public announcements of new vulnerabilities without directly involving MITRE in the details of the specific vulnerabilities. Alternatively, MITRE assigns candidates to issues that have already been publicised (e.g. on Bugtraq or in a security advisory), but that have not been assigned a number by a CNA. The Board members review the candidate, revise it if needed and vote on whether or not it should be included in the CVE list. If a candidate is not accepted, the reason for rejection is recorded on the CVE website. A candidate that is accepted is added as an official entry to the CVE list and published on the website.

In general it takes about one month to assign a CVE number to a candidate, and between six months and a year for the typical candidate to become an official CVE entry. The review process by the Editorial Board allows a minimum of two weeks from proposal to final acceptance and conversion to an official CVE entry. This lengthy review process has pros and cons. The pros are that the process is more controlled and vulnerabilities are scrutinised, and thus the results are more accurate. The cons are that the process takes too long, so vulnerabilities cannot be added to the CVE efficiently; this may result in the security community not being aware of a vulnerability, and could expose systems to attack.

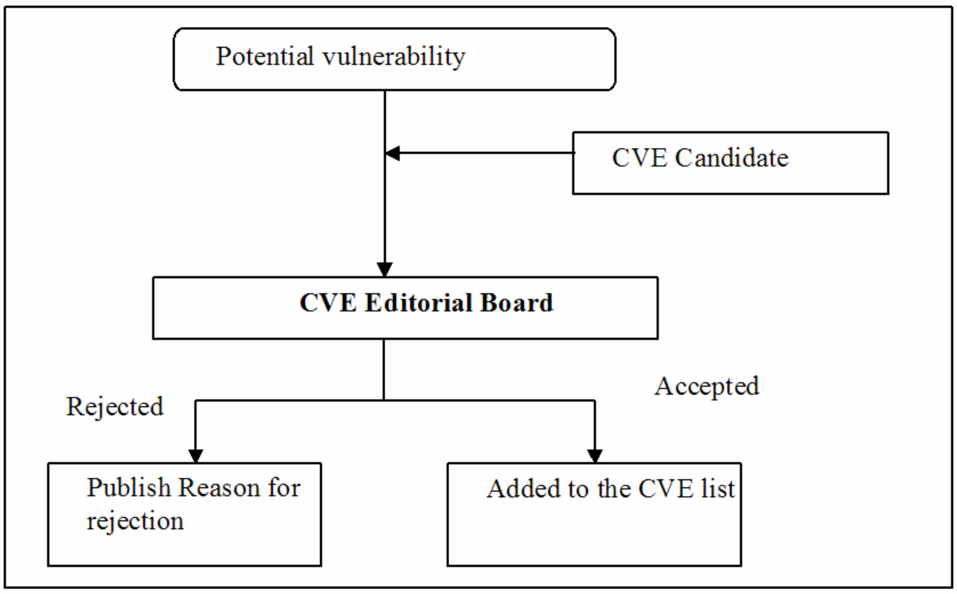


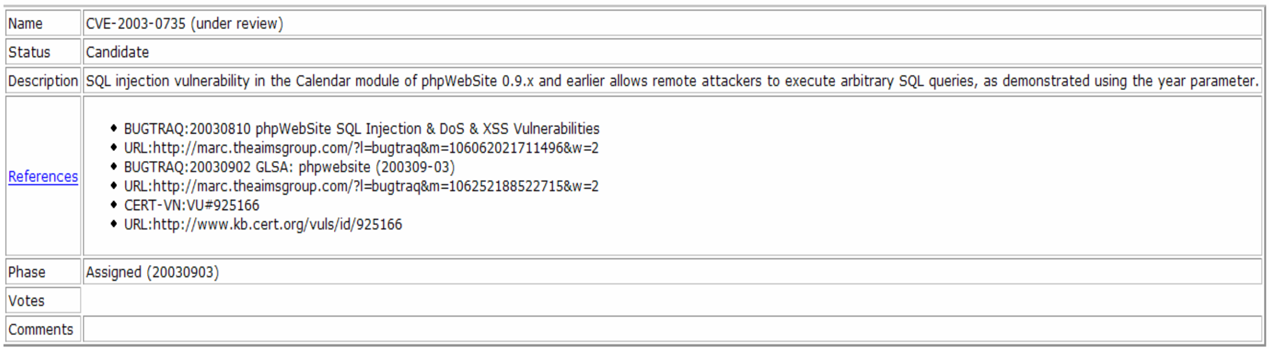
***Figure 3.9 CVE Naming Process***

CVE candidates also have CVE numbers allocated to them, but their status remains that of Candidate. Similar to CVE entries, each candidate has a Description and References field as well.

The following section explains the role of CVE in the general security community.

## 3.7 ROLE OF CVE

The CVE entry for the particular vulnerability illustrated in Figure 3.1, is shown in Figure 3.10 [CVEN 06].



| | |
|---|---|
| Name | CVE-2003-0735 (under review) |
| Status | Candidate |
| Description | SQL injection vulnerability in the Calendar module of phpWebSite 0.9.x and earlier allows remote attackers to execute arbitrary SQL queries, as demonstrated using the year parameter. |
| References | • BUGTRAQ:20030810 phpWebSite SQL Injection & DoS & XSS Vulnerabilities<br>• URL:http://marc.theaimsgroup.com/?l=bugtraq&m=106062021711496&w=2<br>• BUGTRAQ:20030902 GLSA: phpwebsite (200309-03)<br>• URL:http://marc.theaimsgroup.com/?l=bugtraq&m=106252188522715&w=2<br>• CERT-VN:VU#925166<br>• URL:http://www.kb.cert.org/vuls/id/925166 |
| Phase | Assigned (20030903) |
| Votes | |
| Comments | |

***Figure 3.10 CVE Entry 2003-0735***

BugTraq, US-CERT and OSVDB reference the CVE, and the CVE in turn references all these vulnerability databases. This proves the concept that CVE is an intermediate bridge between the vulnerability databases – it maps vulnerabilities across different vulnerability databases and resolves the naming disparity among vulnerability databases.

The CVE serves a critical role in security communities. Security products that make use of it are more interoperable because of the CVE's uses for information sharing. This enhances the communication and security of enterprises using those products. In general, CVE facilitates [BAKE 99]

- communications among users by providing standardised names for reference purposes and a comprehensive vulnerability listing;
- tool interoperability, both immediately as early adopters develop private tool integration strategies and inherently as vendors adopt CVE, and
- tool assessment of vulnerabilities tools.

Certain aspects of tool comparison and assessment are not addressed by CVE. These include [BAKE 99]

- the severity of the vulnerabilities;
- the classification of the vulnerabilities;
- security policies, local configurations, risk postures and related issues, and
- the performance and efficiency of security tools.

## 3.8 CONCLUSION

Many vulnerability databases are available on the market; some are public, such as OSVDB, and some are proprietary to a specific security tool, such as Nessus's Plugins. Vulnerability databases define their own naming schemes and vulnerability classification, with the result that collaboration and interoperation are limited by the disparities among them. CVE is aimed at solving the naming disparity by providing a list of common names for publicly known vulnerabilities and exposures. It is the key to information sharing among various vulnerability scanners. CVE achieves interoperability among vulnerability scanners to improve security coverage and to provide a baseline for evaluating the coverage of various security tools and databases. This can help to determine the appropriate tools that are effective for the organisation's needs.

Although CVE solves the naming disparity, it does not provide a solution for vulnerability classifications. There is no standard categorisation for vulnerabilities. Some organisations such as Nessus, OSVDB, BugTraq define their own classifications, while others (e.g. US-CERT) do not define any classification. From the vulnerability databases discussed in this chapter, it is clear that they classify vulnerabilities independently, and a huge disparity exists with regard to their classification conventions. For example, US-CERT Vulnerability Notes and CVE do not classify vulnerabilities at all. OSVDB, BugTraq and Nessus classify vulnerabilities according to their own classification scheme, and there is no correlation between them. It is essential that classification disparity be resolved, as standardised vulnerability categories can greatly enhance the interoperability, tool comparison and evaluation of vulnerability products. The benefits of standardising vulnerability categories are discussed in the later chapters.

The current research focuses on providing a standard set of vulnerability categories through the use of a data-clustering tool. The next chapter provides an overview of data clustering and discusses a number of different clustering tools. The reason for using data clustering is also discussed in the next chapter.

# CHAPTER 4

# 4 OVERVIEW OF DATA CLUSTERING AND ITS TOOLS

## *4.1 INTRODUCTION*

Data clustering is a method by means of which objects that are somehow similar in characteristics [ALIR 04] are clustered together. In other words, objects that are logically similar are physically grouped together in a cluster. A cluster is therefore a collection of objects that display a strong similarity and are dissimilar to the objects belonging to other clusters. To elaborate further on the concept – consider the following example of a library system. A library has a large variety of books. Books are always kept in the form of clusters. The books that have some kind of similarity among them are placed in one cluster. For example, books on databases are kept on one shelf and books on operating systems are kept on another. Both shelves belong to the same cupboard as they have a similar field: Computers. The shelf and the cupboards are labelled with the relative name and number. If this process of classifying is applied to all books in the library, a user looking for a book on a specific kind of topic would only have to go to that particular cupboard, check the relative shelf and search for the book rather than browse through the entire library. The same concept applies to vulnerabilities. Similar-featured ones are clustered together so that they can be more searchable when classified.

The goal of clustering is to determine the intrinsic grouping in a set of unlabelled data [ALIR 04]. The set of data can be represented as a set of *i*-dimensional vectors. Each *i*-dimensional vector is referred to as a pattern. The set of data (or data set) that is to be clustered is referred to as a data space or problem space. The problem space is typically not uniformly occupied for a large set of multi-dimensional data points. Sparse and crowded areas in the problem space are identified through data clustering, implying that the overall distribution of patterns in the data set is identified as well. This is illustrated by a simple graphical example in Figure 4.1.

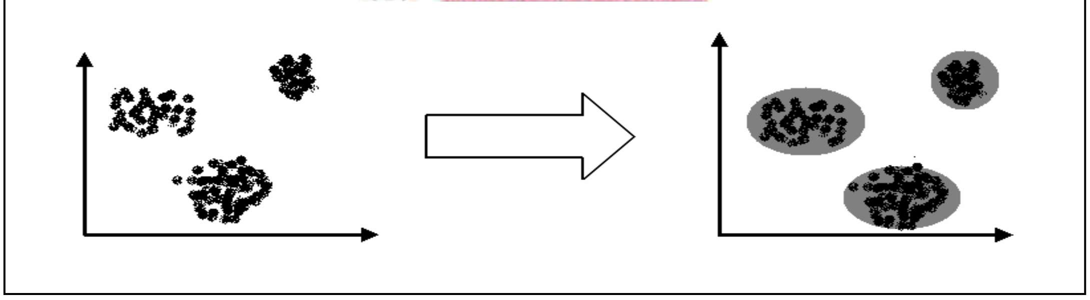


*Figure 4.1 Data Clustering*

The rest of Chapter 4 is divided into seven sections. The first of these explains the motivation for using data clustering to categorise vulnerabilities, followed by a section that outlines the steps in a typical data-clustering process. A brief overview is provided of different data-clustering techniques, as well as of some applications of data clustering. The subsequent section explains the clustering tool selected for this research – the Self-Organising Feature Map (SOM) – in detail, followed by a section that lists the benefits of using SOM in this research. A number of concluding remarks round off this chapter.

## 4.2 MOTIVATION

In this research, data clustering is used because it groups objects that have similar characteristics together into a cluster so that intra-class similarity is maximised and inter-class similarity is minimised. Intra-class similarity refers to the degree of similarity within a cluster and inter-class similarity is the degree of similarity among objects from different clusters. Grouping similar vulnerabilities together in their own categories is ideal for creating vulnerability categories. Another benefit of using data clustering in vulnerability categories is that minimal prior knowledge and assumptions are required and the number of categories does not have to be predetermined, as the data-clustering technique will discover the hidden knowledge in the data set. This makes the clustering approach simpler than normal classification approaches where categories are identified first and each vulnerability is then assigned to the most appropriate category.

Clustering is often confused with classification. Classification, also known as supervised classification, assigns objects to predefined classes, whereas in clustering (unsupervised classification) there are no predefined classes. Those classes are yet to be defined by clustering.

Classification uses supervised learning. Supervised learning is a technique that defines a function that approximates the values in the problem space. The classification technique defines the function by training on a data set that has already been classified. This data set is referred to as the training data. The supervised learning technique repetitively trains on the training data set until the function is able to correctly predict (given a small margin for error) the classification of the training data. The task of the supervised learning technique is to generalise from the training data to untrained data (i.e. data that has not been presented during training) in a reasonable way [JAIN 99]. In essence, classification gets a collection of labelled (pre-classified) patterns and the goal is to label patterns that have not been labelled. Thus, classification requires prior knowledge of classification labels and a pre-classified training data set, which makes it undesirable for this research. Furthermore, the result of the classification might be biased towards the person(s) who did the classification as they need to pre-determine the classification labels and pre-classify a set of vulnerabilities based on their own judgement.

Clustering uses unsupervised learning. It is distinguished from supervised learning by the fact that the training data is not classified. In the case of clustering, the goal is to group the unlabelled set of training data into meaningful classes. Classes are similar to labels, but classes are solely obtained from the data [JAIN 99], as opposed to classification. Clustering requires neither prior knowledge of classification labels nor a pre-classified training data set.

This research focuses on clustering based on unsupervised learning, because it exhibits the following interesting properties [ZENE 04]:

- **Outlier detection**: Patterns that deviate from the normal patterns in the data set are referred to as outliers. The detection of outliers is necessary when categorising vulnerabilities written in natural language. Some vulnerability descriptions can contain very specific words. For example, specific version numbers of software, or specific command parameters like -path or -mode. These can be considered as outliers. Too many outliers can complicate and slow down the clustering process. Thus, clustering methods can be configured to detect and ignore outliers to produce better results.
- **Generalisation**: Unsupervised learning techniques are quite robust and have the ability to assign unseen data to appropriate clusters. New vulnerabilities are being discovered daily, and it is important for clustering methods to correctly cluster these new vulnerabilities in a consistent way.

- **Unsupervised learning**: Minimal expert knowledge is needed and users require no *a priori* knowledge. As discussed earlier in this session, no pre-determined category information is needed in the training data, so the training is not biased.
- **Adaptation**: The input parameters can be adjusted during the training process to optimise the result. For example, the number of map units of a SOM can grow during the training process if needed. Growing SOM is explained in more detail in Section 4.6.

The next section shows the various steps in data clustering.

## 4.3 STEPS FOR DATA CLUSTERING

A typical clustering process involves the following steps [JAIN 99]:

- Pattern representation
- Defining an appropriate pattern proximity measure for the data domain
- Clustering
- Data abstraction (if needed)
- Output assessment (if needed)

Figure 4.2 illustrates the typical steps in a clustering process. It also includes a feedback path where the output of one clustering process can affect subsequent feature extraction and similarity computations [JAIN 99].

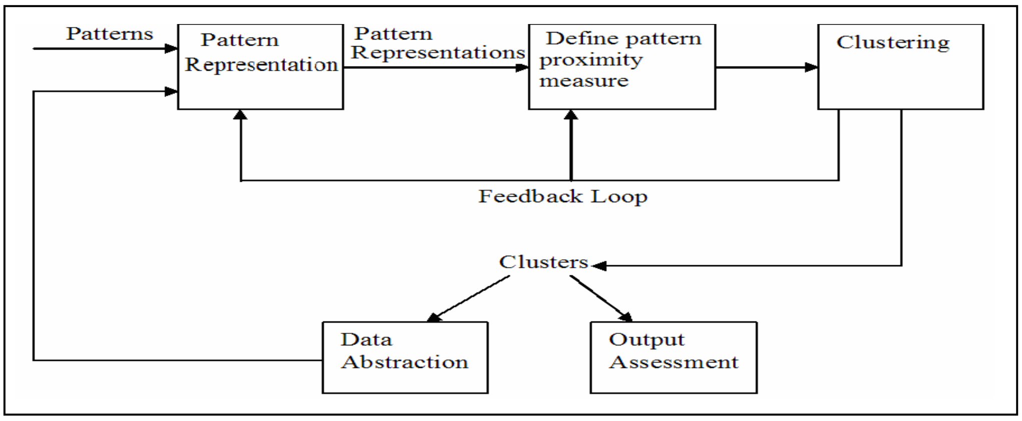


*Figure 4.2 Steps in a Clustering Process*

A pattern, also referred to as a *i*-dimensional feature vector, is a single data item that is used by the clustering algorithm:

$$\mathbf{X} = (X_1, X_2, \ldots\ldots\ldots\ldots\ldots\, X_d),$$

where components $X_i$ of a pattern **X** are called features (or attributes).

The first step in the data-clustering process is **pattern representation**. It refers to the type, scale and number of the features available, the number of classes, and the number of available patterns for the clustering algorithm. It can optionally include feature extraction and/or selection. The use of one or more transformations of the input features to produce new salient features is known as feature extraction. Feature selection selects a subset of the original features to use in clustering, as certain features are unique for all patterns and are therefore not useful for classification. An appropriate set of features can be obtained by using a combination of feature extraction and feature selection for clustering. For example, the original input data set of a clustering algorithm contains *5*-dimensional numeric vectors, where

- the first dimension represents a timestamp;
- the second dimension represents an ID;
- the third dimension represents height;
- the fourth dimension represents width, and
- and the fifth dimension represents weight.

Only three of the five features are important to the clustering algorithm, that is the width, height and weight, as the timestamp does not offer any useful information and the ID is unique for each input vector. Thus, the resulting input patterns after pattern representation are *three*-dimensional numeric vectors, for example (2,3,6) where 2 is the value for the height, 3 is the value for the width and 6 is the value for the weight attribute.

The second step is to **define an appropriate pattern proximity measure for the data domain**. A function that defines the distance on a pair of patterns is used to measure a pattern's proximity to another pattern. Various communities have defined a number of distance functions. A simple distance measure can be used to show dissimilarity between patterns, whereas other similarity measures can be used to characterise the conceptual similarity between patterns [MICH 83]. The most popular distance function is the Euclidean distance. Euclidean distance calculates the dissimilarity between two numerical patterns.

The **clustering** step is also known as the grouping step. It represents the organisation of patterns into clusters, based on pattern similarity. The next section explains various clustering methods that are available today. The choice of a particular method will depend on the type of output desired, the known performance of the method with particular data types, the hardware and software facilities available, and the size of the data set.

A simple and compact representation of a data set is obtained through **data abstraction**. This simplicity is gained in the automatic analysis that a machine can perform efficiently, or it is illustrated in a representation that is easy to comprehend and intuitively appealing (usually a two-dimensional illustration of the clusters). In the clustering context, a typical data abstraction is a compact description of each cluster in terms of cluster prototypes or representative patterns such as the centroid [DIDA 76]. Informally, the centroid is a representation of the patterns that belong to the same cluster. An example of data abstraction is to list, instead of all the vulnerabilities belonging to a particular cluster, only those vulnerabilities mapped to the centroids of the cluster. Less data is presented and the key characteristics of the cluster are highlighted. Clustering methods have shown to be ideally suited to achieving data abstraction.

Once the clusters are generated, they can be assessed if needed. The **output assessment** step is also known as cluster validity analysis; it is performed to determine whether the patterns are classified in a meaningful context. In this research, after the vulnerability categories are generated by the clustering algorithm, the categories are internally examined against a list of defined evaluation criteria. The categories are also examined in relation to other vulnerability categories defined by other parties. Details of this evaluation will be discussed in a later chapter.

The data-clustering process has been explained in a nutshell here and the next section will discuss various clustering methods that are available today.

## *4.4 TAXONOMY OF DATA-CLUSTERING METHODS*

Data-clustering techniques have been studied extensively in statistics, machine learning and data mining with many methods proposed and studied. Major clustering techniques can be classified into the following five categories [HANJ 00]:

- Partitioning algorithms

- Hierarchical algorithms
- Density-based methods
- Grid-based methods
- Model-based methods

Each category is explained in more detail in the subsections that follow.

### 4.4.1 Partitioning Algorithms

Partitioning algorithms construct a partition of a database X containing *n* objects into a set of *k* clusters and then evaluate them by specific criteria. The most classical and popular partitioning algorithms are k-means [MACQ 67] and k-medoid [KAUF 90]. In k-means, each cluster is represented by the centroid of that particular cluster, whereas in the k-medoid method, the clusters are represented by the "central" objects of the clusters. Once cluster representatives are selected, data points are assigned to these representatives, and iteratively, new and better representatives are selected and points reassigned until no better representatives can be found. All the partitioning algorithms have a similar clustering quality and the major difficulties with these methods include the following:

- The *k* number of clusters needs to be found and known prior to clustering, which requires at least some prior domain knowledge for the problem at hand, which is often not available.
- It is difficult to identify clusters with large variations in size; large genuine clusters tend to be split.

### 4.4.2 Hierarchical Algorithms

Hierarchical clustering algorithms produce a representation of the nested grouping relationship among objects, using a distance matrix as the criterion. There are two types of hierarchical algorithms [OSMA 05]:

- **Agglomerative**: The clustering hierarchy is formed from the bottom up. At the start each data object is a cluster by itself, and then small clusters are merged into bigger clusters at each level of the hierarchy until at the top of the hierarchy all the data objects are in one cluster. An example of an agglomerative algorithm is AGNES (Agglomerative Nesting) [KAUF 90].

- **Divisive**: The inverse of agglomerative. It does the reverse by starting with all objects in one cluster and subdividing them into smaller clusters. However, the divisive methods are not generally available and rarely have been applied. An example of such an algorithm is DIANA (Divisive Analysis) [KAUF 90].

The number of clusters, *k*, is not required as an input for hierarchical clustering algorithms; however, they do need a termination condition. They also do not scale well due to high computational complexity of at least O ($n^2$) where *n* is the number of input objects.

### 4.4.3 Density-based Methods

For density-based clustering methods, clustering is based on density and connectivity functions. The advantages of density-based clustering methods are that they can discover clusters with arbitrary shapes and they do not need to preset the number of clusters. They can also handle noises in the data set. Noises are inputs with missing values, outliers, etc. Examples of density-based clustering methods are DBSCAN (Density Based Spatial Clustering of Applications with Noise) [EAST 96] and OPTICS (Ordering Points To Identify the Clustering Structure) [ANKE 99].

### 4.4.4 Grid-based Methods

Grid-based methods first quantise the clustering space into a finite number of cells, and then perform clustering on the grid cells. They use a multi-resolution grid data structure. The main advantage of grid-based methods is that their speed only depends on the resolution of grid, but not on the size of the data set. Thus they are more suitable for use in high density data sets with a huge number of data objects in limited space. Popular grid-based methods are WaveCluster [SHEI 98] and CLIQUE (Clustering in QUEst) [AGRA 98].

### 4.4.5 Model-based Methods

Lastly, the model-based clustering methods use certain models for clusters and strive to optimise the fit between the data and the model. They adopt three different approaches, namely the neural network approach, the probability density-based approach and the statistical approach. The Self-Organising Feature Map (SOM) is the best known neural network approach to clustering. It can be viewed as a non-linear projection from an *i*-dimensional input space onto a lower order (typically two-dimensional) regular lattice of cells. Such a mapping is used to identify clusters of elements that are similar in terms of Euclidean distance in the original space. SOM is a feed-forward neural network approach that uses an unsupervised training algorithm and, through a process called self-organisation, configures the output units into a topological representation of the original data. This will be explained in more detail in the later section as SOM is the tool that we have selected for our research.

The probability density-based approach groups data based on the probability density model. It is based on how many (possibly weighted) features/attributes are the same. The COBWEB [FISH 89] method is an example of such an approach. The statistical approach clusters data based on statistical models like the Gaussian distributions. The Gaussian mixture model [BANF 93] is a probabilistic variant of a k-means method. It starts by choosing seeds and regarding the seeds as means of Gaussian distributions; then iterates until the Gaussians are no longer moving.

Clustering algorithms have been used in a large variety of applications. In the science of computers, people make use of data clustering, especially in the field of information retrieval. Data clustering has been successfully used in the field of image segmentation, pattern recognitions, computational biology and bioinformatics, and data mining [MURT 95].

The major clustering methods were briefly discussed in this section. The following section explains the SOM in more detail as it is the clustering tool that has been selected for this research. Reasons for selecting the SOM tool are also explained in the subsequent section.

## 4.5 SELF-ORGANISING FEATURE MAP (SOM)

The self-organising feature map (SOM) is a method of unsupervised learning, based on a grid of artificial neuron units whose weights are adapted to match input vectors in a training set. It was first described by the Finnish professor Teuvo Kohonen and is thus sometimes referred to as a Kohonen map [KOHO 95].

The SOM algorithm is fed with *i*-dimensional feature vectors, where *i* represents the number of dimensions. In most applications, however, the number of dimensions will be high. Output maps can also be generated in different dimensions (*1*-dimensional, *2*-dimensional, etc.), but most popular are two- and three-dimensional maps because SOMs are mainly used to reduce the problem space into a two- or three-dimensional space that is more comprehensible to humans.

SOM is a clustering tool that is useful for visualising high-dimensional data in two-dimensional space [ENGE 02]. The benefits of using SOM will be discussed in the section that follows. It consists of a map of artificial neuron units and a set of *i*-dimensional input vectors known as the training set (see Figure 4.3), where the circles in the map symbolise the neuron units. In this figure, it is a 5x6 map that contains 30 neuron units. Each unit is a weight vector that has the same dimension as the input vectors. For example, if a SOM takes a *5*-dimensional input vector, the neuron units in the SOM will also be a *5*-dimensional vector representing the weights of that unit.

The following is a summary of the SOM algorithm [KOHO 95]:

- Randomise/Initialise the weight vectors of the map's units.
- Read an input vector.
- Traverse each unit in the map:
    - Use the Euclidean distance formula to calculate the distance between the input vector and the unit.
    - Track the node that produces the smallest distance (this node will be called the Best Match Unit or BMU).
- Update the units in the neighbourhood of BMU by adjusting their weights so that they can be closer to the input vector.

- Repeat the process until the termination condition is met.

The termination condition can be the maximum number of times or epochs to repeat the above-mentioned process, or if the calculated error is less than a certain value. This error is the map quantisation error, which is calculated after each training iteration. The error describes the accuracy or "how good" the map is trained, with smaller values indicating a better result. After training, similar patterns can be mapped to units that are close together in the SOM.

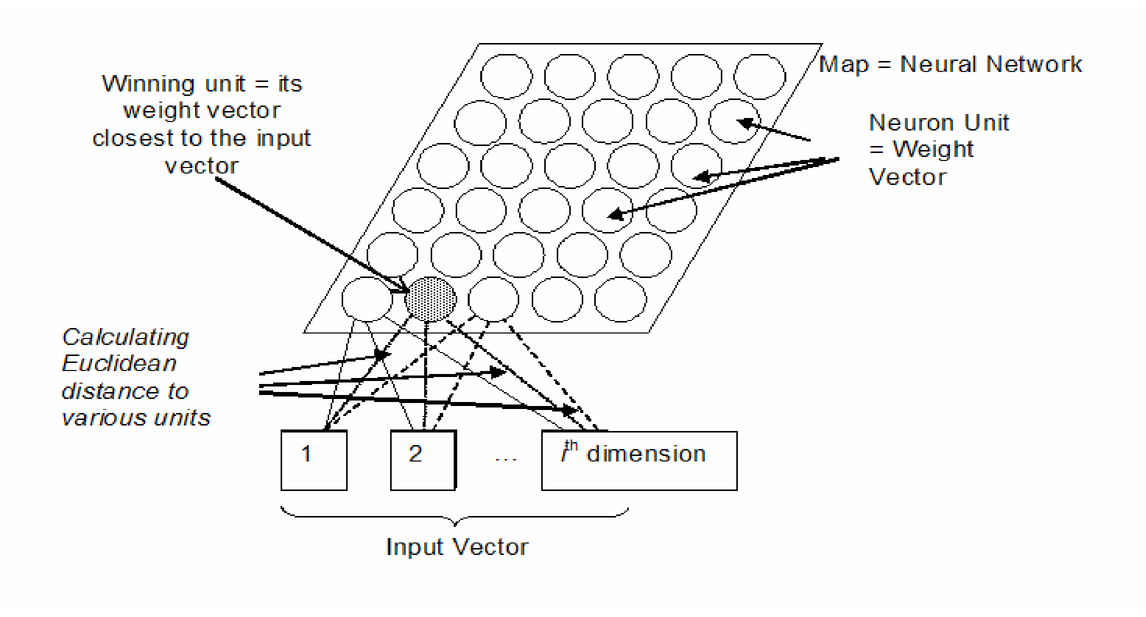

Winning unit = its weight vector closest to the input vector
Map = Neural Network
Neuron Unit = Weight Vector
Calculating Euclidean distance to various units
1
2
...
$i^{th}$ dimension
Input Vector

*Figure 4.3 SOM Architecture*

Using Figure 4.3, the SOM algorithm can be easily explained in terms of a set of artificial neuron units – each having its own physical location on the output map – that take part in a winner-take-all process (competitive learning strategy [ANGE 93]). A unit with its weight vector closest to the vector of inputs (according to the Euclidean distance) is declared the winner and its weights are adjusted making them closer to the input vector. Each node has a set of neighbours. The weights of the defined neighbours of the winning unit are also adjusted to a lesser degree. The further the neighbour is from the winning unit, the less the weight is adjusted. This process is then repeated for each input vector for a number of training iterations. Different inputs produce different winners. The SOM associates units with groups or patterns in the input data set. For labelled patterns, the labels can be attached to the associated units in the trained network.

Figure 4.4 shows a simplified example of the SOM training process. The SOM map in this example has 3 rows and 3 columns, and thus it forms a 3x3 SOM map. The input vector is 4-

dimensional, thus the weight vectors for the SOM units is also 4-dimensional. Each of the units on the map is a specific weight vector as it is shown on the diagram. For the input vector (2, 2, 4, 4), its Euclidean distance to all of the units is calculated and the winning unit is selected as that unit that is closest to the input vector. For example, the Euclidean distance to the winning unit in Figure 4.4 is calculated as follows:

$$\textit{distance} = ((2-1)^2 + (2-2)^2 + (4-3)^2 + (4-4)^2)^{0.5} = 1.41421$$

After the winning unit is found, the weights are adjusted. The weight vector of the winning unit is adjusted to be even closer to the input vector, and its neighbouring units' weight vectors are also adjusted to be closer to that of the winning unit. This, in turn, results in the neighbouring units moving further away from other units in the map because they are moving closer to other units on the map. Gradually, clusters will be formed in the SOM in this way.

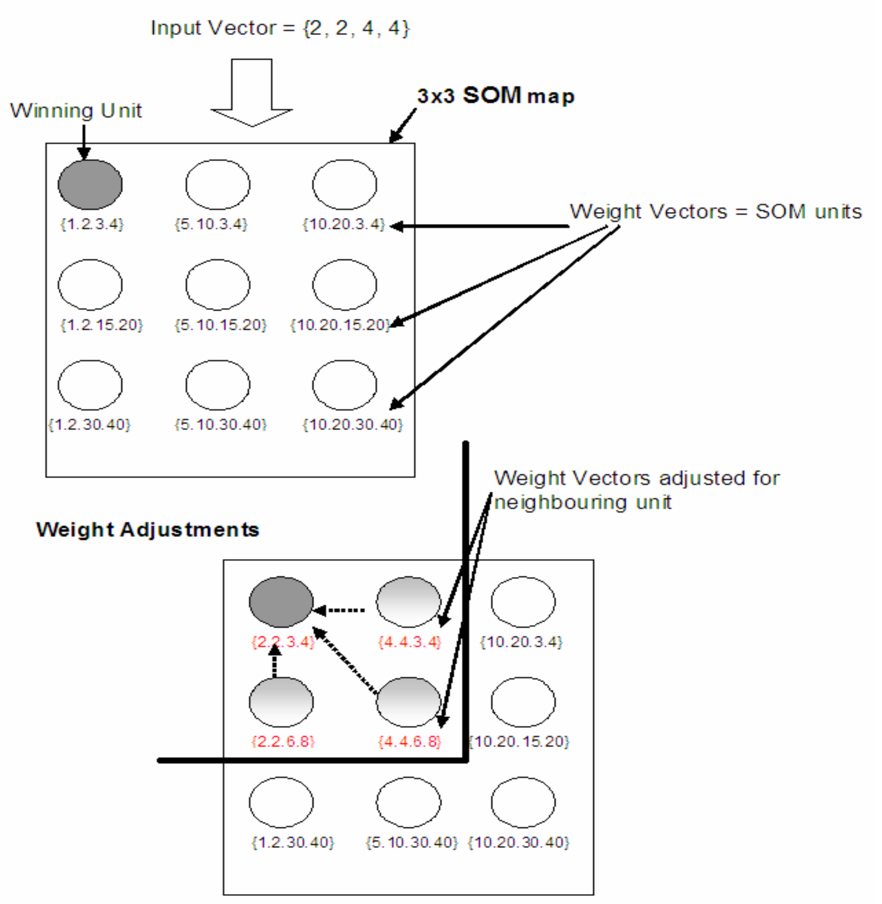


***Figure 4.4 SOM Training Process Example***

Before using the SOM tool, the dimensions of the map need to be determined. The dimensions are measured by the number of units (e.g. a 6x5 map has 6 rows of 5 units each). Usually the number of units in the SOM is less than the size of the data set. For example, the data set for this research contains 3 052 input vectors but the SOM only needs less than 400 units (20x20 map). If too many units are used, the SOM will overfit the input vectors; in addition, it may also cause undesirably small clusters. Overfitting implies that the SOM is memorising the training set and will not be able to correctly classify input vectors that have not originally been in the training data set. Too many units may have the additional side-effect of causing many units to have a zero or close to zero frequency, even though proper clusters have been formed. The frequency of a neuron unit is the number of input vectors for which that unit has been selected as the winning unit. Alternatively, too few units result in fewer clusters and a high variance among the cluster members, which do not provide the optimal categories of the training vectors. This dilemma is resolved using a Growing SOM [KIRK 01], in which the SOM architecture attempts to adapt to the training set. The Growing SOM first starts with a small number of units and then adapts by adding more units as needed, resulting in an architecture that is best suited for the training set. The map-growing process coexists with the training process.

In this research, the Growing SOM is used. Figure 4.5 illustrates the growing process. The following is a summary of the Growing SOM algorithm [ENGE 02]:

- Initialise the units' weight vectors for a small, undersized SOM.
- Grow the map:
    - Train the SOM for *n* input vectors, using any SOM training method.
    - Find the unit *ij* with the largest training error, which is the sum of all errors for all input patterns.
    - Find the unit *rx* from the immediate neighbours in the row-dimension of the map that has the largest Euclidean distance to unit *ij*, and the immediate neighbour unit *cd* in the column-dimension with the furthest Euclidean distance.
    - Insert a row between unit *ij* and *rx* and a column between unit *ij* and *cd*.
    - For each unit *mn* in the new column, initialise the corresponding weight vector $\vec{w}_{mn} = \alpha(\vec{w}_{m-1,n} + \vec{w}_{m+1,n})$ and for each unit in the new row, $\vec{w}_{mn} = \alpha(\vec{w}_{m,n-1} + \vec{w}_{m,n+1})$, where $\alpha \in (0,1)$. This step is called the interpolation step. The idea of the interpolation step is to assign a weight vector to the new unit so that it reduces the error of the largest error unit *ij* by removing input vectors from the unit *ij*. Thus 0.5 is a good value for $\alpha$. A value larger than that will position unit *mn* further apart from *ij*, which may result in fewer input vectors being removed from unit *ij*. A value smaller than 0.5 will have the opposite effect. Thus a new column unit will initialise its weight vector to be half of the sum of the

weights of the unit to the left and the unit to the right of it. A new row unit will initialise its weight vector to be half of the sum of the weights of the unit above it and the unit below it.

- o Stop growing when any one of the following conditions are met:
  - The maximum map size has been reached. The maximum map size should be determined with care. The upper bound on the maximum map size is simply the number of input vectors in the training set. However, this is undesirable because it may cause overfitting. If it is too small, the SOM may not converge to the required error.
  - The largest unit error is less than a user-specified error threshold, $\lambda$. The error threshold $\lambda$ is important to ensure that a sufficient map size is constructed. A small value for $\lambda$ may result in large map architecture, while a large $\lambda$ may result in longer training times to reach a large enough architecture.
  - The map has converged to the specified error.

- Refine the weights of the final SOM architecture with additional training steps until convergence has been reached.

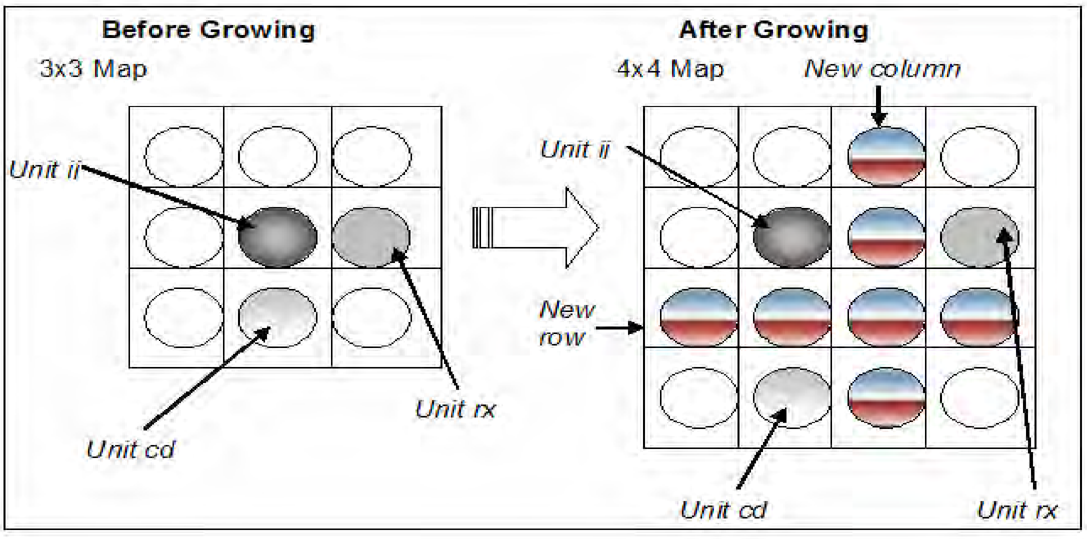


*Figure 4.5 Growing the SOM Map*

Like most artificial neural networks, the SOM has two modes of operation:

- **Training Phase –** during the training phase, a map is built. The neural network organises itself, using a competitive process. The network must be given a large number of input vectors, representing the kind of vectors that are expected during the mapping phase (if any). Otherwise, all input vectors must be administered several times.

- **Mapping Phase –** this phase happens after the training phase has been completed. During the mapping phase, a new input vector may quickly be given a location on the map. It is automatically classified or categorised by the trained SOM. There will be one single winning unit, namely the unit whose weight vector lies closest to the input vector.

In this research, the training phase mode categorises the CVE list, and the mapping phase mode classifies new unseen vulnerabilities that are not in the training data. The categorisation process is discussed in greater detail in the next chapter. The following section explains the reasons for and benefits of using the SOM for this research.

## *4.6 BENEFITS OF USING THE SOM FOR THIS RESEARCH PROJECT*

Below is a list of the reasons for selecting the SOM as the clustering tool for this research. The reasons are discussed in depth in the paragraphs that follow.

- SOM is unsupervised learning, therefore no prior knowledge of the problem is required.
- SOM is highly scalable.
- SOM is able to deal with noise and outliers.
- SOM is insensitive to the order of input records.
- SOM can handle high dimensionality.
- SOM aids in interpretability and usability.
- SOM aids in visualisation.

The SOM is selected because it is an unsupervised learning neural network [DESA 94], which means that no target solution is needed beforehand. As explained in Section 4.2, no prior knowledge needed for the vulnerabilities and vulnerability categories implies that the end result is not biased. The SOM will try to discover the pattern or knowledge hidden in the input data set. The ability to handle high dimensional space is very important to this research as the researcher is trying to cluster natural language descriptions. Natural language has approximately a million words [LING 06], thus the dimensions of the data set will be huge. The input patterns can be in any random order, and SOM is able to deal with noise, outliers and missing values in the data set [KOHO 95]. As shown in Section 4.2, this is particularly important when dealing with natural language descriptions. The SOM is also scalable. This feature is greatly desired because, after the training process, a new pattern can quickly be given a location on the map and it is automatically classified or categorised. There will be one single winning unit: the unit whose weight vector lies closest to the input vector.

The results obtained from the SOM are easy to visualise and interpret [ENGE 02]. This helps when inspecting the categories that are formed. Figure 4.6 shows an example of a cluster map. This map reveals eight different clusters represented by different colours. The clusters differ in size.

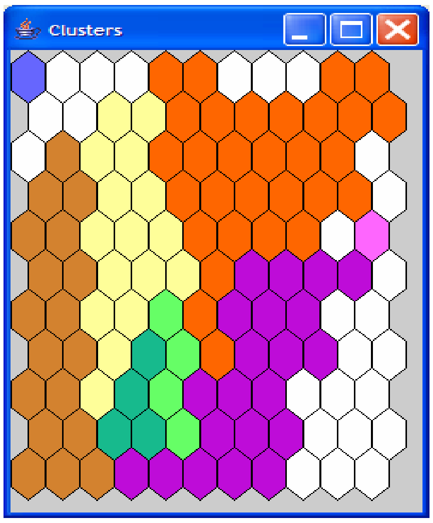


*Figure 4.6 Cluster Map*

SOM clusters vulnerabilities into different categories of a similar nature. However, these clusters are not labelled and this needs to be done manually by careful inspection of the cluster's properties. The labelling procedure is performed only once. When completed, a new entry can be mapped to the SOM and classified easily. This has no serious impact, as the clusters have to be inspected manually to discover what caused the SOM to form the cluster.

## 4.7 CONCLUSION

Data clustering is a common technique used for data analysis, which has found applications in many fields including machine learning, pattern recognition, image analysis, bioinformatics and data mining. This chapter explained the basic concept of data clustering. Data clustering uses unsupervised learning algorithms, whereas supervised learning is required for data classification. These two types of learning algorithms and the differences between them were briefly discussed. Motivations for using data clustering were provided in Section 4.2.

This chapter has also discussed the various steps in clustering: (1) pattern representation, (2) similarity computation, (3) clustering process, (4) data abstraction, and (5) output assessment. Various clustering tools and approaches were looked at. The Self-Organising Feature Map (SOM) is an example of a model-based clustering method using an unsupervised neural network approach. In contrast to the classic clustering methods, a SOM provides easy visualisation and imposes few assumptions, restrictions and prior knowledge. It also handles large data sets to detect patterns and structures in the data. For this research, SOM is selected as the clustering tool and a list of the reasons for this decision was specified in Section 4.6.

The most important step before clustering using the SOM is the data preprocessing. The SOM uses a set of *i*-dimensional training vectors. Thus it is not possible to parse the natural language descriptions in the CVE directly to the SOM tool, because the descriptions vary in length and are not of a numeric type. The next chapter discusses the necessary data preprocessing to transform the data format of the CVE repository into a format that can be used and processed by the SOM. It will also explain the categorisation process in detail.

# CHAPTER 5

# 5 CONSTRUCTING STANDARDISED VULNERABILITY CATEGORIES

## 5.1 INTRODUCTION

Previous chapters have discussed the need for standardising vulnerability categories, the CVE list and the SOM. Several researchers and institutions have done related work on vulnerability categorisation, and some of them are discussed in this chapter. This chapter explains the entire categorisation process in detail. The SOM is used to categorise the CVE list to construct a standard set of vulnerability categories. The categorisation process is a two-phased approach. The first phase is designed to focus on generating categories by clustering the CVE list, and the second phase is designed to be an on-going process that labels newly discovered vulnerabilities as they get published.

This chapter is organised as follows: Section 5.2 discusses work related to the current research. Section 5.3 explains the two-phased categorisation process. While the motivation for choosing the SOM for this research has already been explained in Section 4.6, this section also discusses the motivation to categorise the CVE list. All the steps in each phase are discussed in detail. Section 5.4 contains some remarks to conclude this chapter.

## 5.2 RELATED WORK

Various researchers have done related work on categorising vulnerabilities. This section discusses some of these studies briefly.

Venter et al. identified a set of 13 Harmonised Vulnerability Categories, which represent the entire range of vulnerabilities that are currently known [VENT 02]. The categories are: Password cracking and sniffing; Network and system information gathering; Backdoors; Trojans and remote controlling; Unauthorised access to remote connections and services; Privilege and user escalation; Spoofing or masquerading; Misconfigurations; Denial-of-Services (DoS) and buffer overflows; Viruses and worms; Hardware specific; Software specific and updates; Security policy violations. This set of vulnerability categories is then used to perform vulnerability assessment.

The Harmonised Vulnerability Categories were identified by analysing the security vulnerabilities found in current literature, current vulnerability scanners and on the Internet. Thus the vulnerability categories are identified based on the knowledge and expertise of the person or group that has to perform the categorisation. Human expertise is required to manually assign Harmonised Vulnerability Categories to all the vulnerabilities. There are thousands of known vulnerabilities already, and many more found on a daily basis, so although it is a once-off assignment, this process is tedious.

Other than the Harmonised Vulnerability Categories, CISCO [CISC 06], which is one of the reputed Network Security companies, also classified main vulnerabilities of systems into five categories. These are Design Errors; Protocol Weaknesses; Software Vulnerabilities; Misconfiguration; Hostile Code [KUJA 03]. Microsoft defined a STRIDE Threat Model that consists of six threat categories: Spoofing Identity; Tempering with Data; Repudiation; Information Disclosure; Denial of Service, Elevation of Privilege [MICR 02]. Many vulnerability scanners also identified their proprietary vulnerability categories. For example, SAINT, which is a popular commercial vulnerability scanner product, identified twelve vulnerability categories: Web; Mail; Ftp; Shell; Print; RPC; DNS; Database; Net; Windows; Passwords; Miscellaneous [SAIN 06]. Missouri Research & Education Network also identifies 25 vulnerability categories [MORE 04]. Table 5.1 shows a summary of the different classification schemes discussed in this section. There is no standard for categorising vulnerabilities or for assessing vulnerability scanners, as mentioned in the problem statement in Chapter 1, and the categorisation process proposed in this research therefore aims to solve this problem by constructing standardised vulnerability categories.

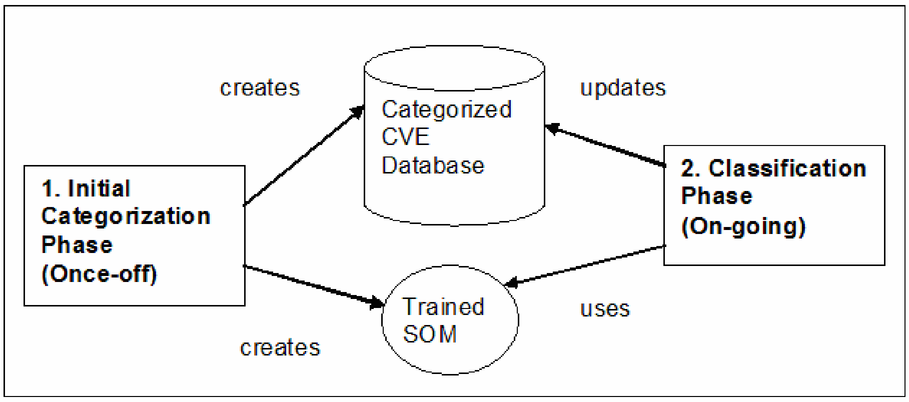
***Table 5.1. A Summary of vulnerability classification schemes discussed***

| **Scheme** | **Author / Organization** | **Number of Categories** |
|---|---|---|
| Harmonised Vulnerability Categories | Venter et al. | 13 |
| CISCO Vulnerability Categories | CISCO | 5 |
| STRIDE Threat Model | Microsoft | 6 |
| Saint Scanner’s vulnerability categories | Saint | 12 |

## 5.3 CATEGORISATION PROCESS

Figure 5.1 outlines the categorisation process. There are two phases in the categorisation process, namely:

1. Initial Categorisation
2. Classification



creates
Categorized CVE Database
updates
1. Initial Categorization Phase (Once-off)
2. Classification Phase (On-going)
Trained SOM
uses
creates

*Figure 5.1 The Categorisation Process*

The Initial Categorisation Phase is executed once, with the sole purpose of creating a standard set of vulnerability categories from the CVE data. The fact that CVE provides an internationally accepted naming standard for common vulnerabilities makes it a more suitable data source to perform clustering than any other vulnerability database. Another reason for selecting CVE as the data source is the nature of its “Description” field. Each CVE entry contains a short and concise description. The descriptions are carefully worded by the CVE Editorial Board, and vulnerabilities of a similar nature are named in a consistent way. Therefore, this description field is perfect for use to perform clustering. CVE is public and widely referenced by other vulnerability databases and tools. According to MITRE, there are currently 42 vulnerability databases (e.g. US-CERT Vulnerability Notes Database [CERT 06]; Cisco Secure Encyclopaedia [CISC 06]), as well as more than 50 vulnerability scanners (e.g. Internet Scanner 6.5 [ISSC 06]; Nessus Security Scanner [NESS 06]; SAINT [SAIN 06]) worldwide that reference or intend to reference the CVE [MITR 06]. Although only a few are actually CVE compatible at the moment, most of them declare CVE output and are CVE searchable. This means that a user can perform a search using a CVE name to find related information and the results presented will include the related CVE name(s). Thus, by categorising CVE entries, the researcher is actually standardising the categorisation of the vulnerabilities across different vulnerability databases and scanners.

Vulnerability categories are created by clustering the CVE data with the use of a SOM. The vulnerability categories are saved in the Categorised CVE Database for reference in the Classification Phase.

The Classification Phase depends on the Initial Categorisation Phase as it uses the fully trained SOM to perform the classification of new vulnerabilities. The newly classified vulnerabilities are saved in the Categorised CVE Database for future reference. Many new vulnerabilities are discovered every day, and without the Classification Phase the categorisation result will be outdated very soon. The following subsections explain the Initial Categorisation and Classification phases in detail.

### 5.3.1 Phase 1: Initial Categorisation

The Initial Categorisation phase is a once-off initialisation of the Categorised CVE Database and SOM. The process is outlined in Figure 5.2. Firstly, the CVE repository is downloaded from the CVE web site and imported to a database; it is used as the data source for this categorisation process. The data preprocessing process transforms the natural language description of each CVE entry into an *i*-dimensional numeric vector that can be used by the SOM. After the transformation, a SOM tool is used to perform clustering on the transformed data set. This is the training phase of the SOM. The intention is to group similar vulnerabilities into their own vulnerability category. Once the SOM has completed training, the clusters are generated, which can then be inspected and labelled. The clusters represent the various categories generated from the CVE Database. The cluster labels are saved in the Categorised CVE database for later use. The result of this initial categorisation phase is the standard set of vulnerability categories and a trained SOM. This process is discussed in detail in the subsections to follow.

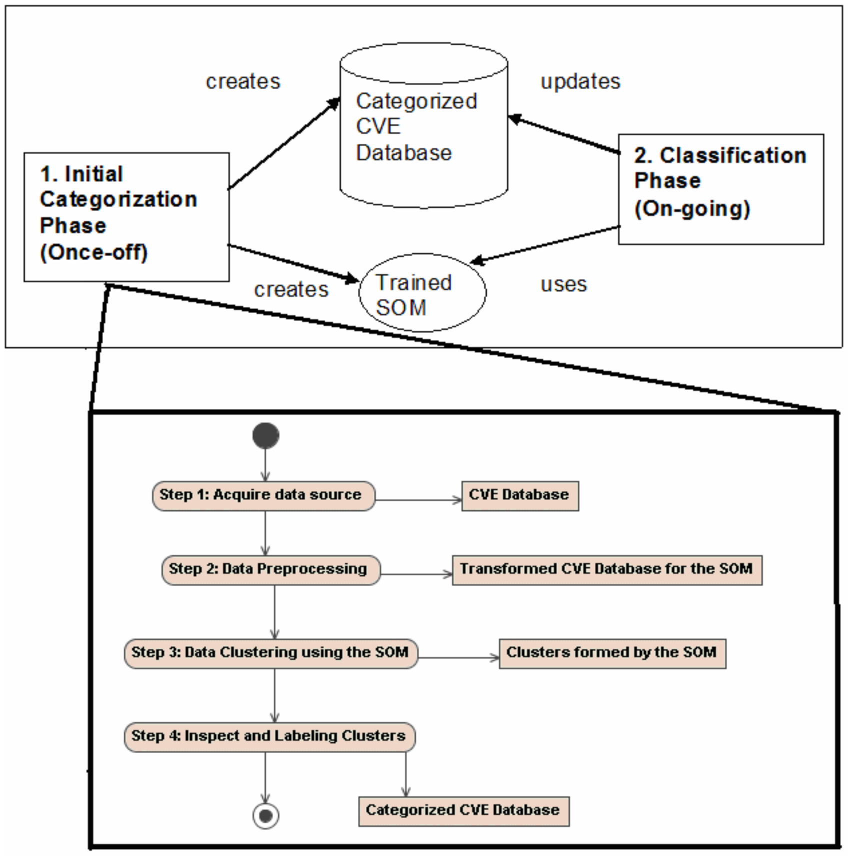

creates
Categorized CVE Database
updates
1. Initial Categorization Phase (Once-off)
2. Classification Phase (On-going)
creates
Trained SOM
uses
Step 1: Acquire data source
CVE Database
Step 2: Data Preprocessing
Transformed CVE Database for the SOM
Step 3: Data Clustering using the SOM
Clusters formed by the SOM
Step 4: Inspect and Labeling Clusters
Categorized CVE Database

*Figure 5.2 Initial Categorisation Phase*

#### 5.3.1.1 Step 1: Acquire data source

The first step of the initial categorisation phase is to acquire the data source. The CVE list can be downloaded from the CVE website [MITR 06] in various formats, such as XML, HTML, Text and Comma-Separated Vector (CSV) format. The CSV format was selected because it can be easily imported into a Microsoft Access Database. The CVE version used for this research is version number 20040901 and it contains a total of 3 052 official entries (excluding CVE candidates with status “Candidate”).

The CVE database table has the exact layout as the CVE list excluding the “Status” field since only the official CVE entries are used in this Initial Categorisation phase. It contains three columns: Key, Description and Reference, where the primary key of the table is the Key. The Description is a short but concise note describing the vulnerability, and the Reference contains other publicly available documents for the particular vulnerability. The following table shows an extract of the CVE table.

*Table 5.2 Extract from CVE database table*

| CVE | | |
|---|---|---|
| **Key** | **Description** | **Reference** |
| CVE-1999-0002 | Buffer overflow in NFS mountd gives root access to remote attackers, mostly in Linux systems. | SGI:19981006-01-I,CERT:CA-98.12.mountd,CIAC:J-006,BID:121,XF:linux-mountd-bo |
| CVE-1999-0003 | Execute commands as root via buffer overflow in Tooltalk database server (rpc.ttdbserverd) | NAI:NAI-29,CERT:CA-98.11.tooltalk,SGI:19981101-01-A,SGI:19981101-01-PX,XF:aix-ttdbserver,XF:tooltalk,BID:122 |
| CVE-1999-0005 | Arbitrary command execution via IMAP buffer overflow in authenticate command. | CERT:CA-98.09.imapd,SUN:00177,BID:130,XF:imap-authenticate-bo |
| CVE-1999-0006 | Buffer overflow in POP servers based on BSD/Qualcomm's qpopper allows remote attackers to gain root access using a long PASS command. | CERT:CA-98.08.qpopper_vul,SGI:19980801-01-I,AUSCERT:AA-98.01,XF:qpopper-pass-overflow,BID:133 |
| CVE-1999-0007 | Information from SSL-encrypted sessions via PKCS #1 | CERT:CA-98.07.PKCS,XF:nt-ssl-fix |
| CVE-1999-0008 | Buffer overflow in NIS+, in Sun's rpc.nisd program | CERT:CA-98.06.nisd,SUN:00170,ISS:June10,1998,XF:nisd-bo-check |
| CVE-1999-0009 | Inverse query buffer overflow in BIND 4.9 and BIND 8 Releases. | SGI:19980603-01-PX,HP:HPSBUX9808-083,SUN:00180,CERT:CA-98.05.bind_problems,XF:bind-bo,BID:134 |

#### 5.3.1.2 Step 2: Data Preprocessing

Data-preprocessing techniques are used to improve the quality of the data and the accuracy and efficiency of the clustering process. It is the most important step before clustering using the SOM, because of the following reasons:

- The SOM can train only on numerical or Boolean vectors.
- The inclusion of features that are not useful for clustering, such as unique timestamps or IDs, will generate clusters that do not offer any useful information.
- Missing data in the data set needs to extrapolated, or removed from the training process.
- Normalising the numeric vectors may improve the accuracy of the SOM, and therefore reduce the training period.

The various forms of data preprocessing [HANK 01] include the following:

- **Data cleaning** fills in missing values, smoothes noisy data, identifies or removes outliers, and resolves inconsistencies.
- **Data integration** involves the inclusion of multiple sources of data (databases, data cubes or files). Integrating many data sources may cause redundancies that will slow down the

clustering process and this preprocessing step must take measures to ensure that such redundancies are removed.

- The **data transformation** step transforms or consolidates data into a form that is appropriate for clustering.
- **Data reduction** obtains a reduced representation of the data set that is much smaller in volume, yet produces almost the same analytical analysis.

If the SOM is trained on inaccurate data, incomprehensible clusters may be generated. In this research, data integration is not necessary as the CVE list is one data file. Data reduction is also not performed since the CVE table is not large in comparison to the size of normal input data sets (it contains only 3 052 entries). Therefore the first step in data preprocessing for this research is data cleaning.

Data cleaning consists of removing insignificant words and punctuation from each entry in the description column of the CVE list. Removing all meaningless words simplifies the training process for the SOM and prevents clustering around those words. The following rules are applied when deciding which words to remove:

- Remove all the preposition words such as *by*, *without*, *when*, *and*, *that*, etc.
- Remove all adjectives, adverbs and verbs.
- Remove all words consisting of a single character.
- Remove all the software version numbers.
- Remove all words that occur less than a threshold number of times across the entire CVE. For this research, the threshold is set to 10.

All the words that can be removed by applying the above rules are saved in a file called InsignificantWords for repeated use. This file is shown in Appendix A.

The accuracy of the SOM-clustering process is not impaired by removing words with a small number of occurrences. Even if those words might appear to be meaningful, their limited occurrence will have minimal influence on clustering. Thus, they can be removed to improve the efficiency of the clustering process. The occurrence threshold value should be less than the expected number of clusters. If the threshold value is too large, it may remove significant words that do have an influence on clustering. A threshold value of 10 is used for this research, as it was found that the median value for the number of occurrences is 12. Therefore, the change in the resulting categories after removing a word that has fewer occurrences than the median is trivial.

The SOM tool is trained on a set of *i*-dimensional numeric vectors. Thus, natural language descriptions in the CVE cannot be directly used by SOM, because the descriptions vary in length and are not of a numeric type. Before performing the clustering process, data transformation is required to transform the descriptions of the CVE entries into fixed-length numerical vectors. To achieve this, a set of significant words, *S*, is created from the description of the entire collection of CVE entries after all insignificant words have been removed. For each entry in the CVE list, a numeric vector is created that represents the occurrence of each word in *S*. For example, if *S* was the set of words {a, b, c, x, y, z}, then an entry such as "a c c x z" will produce a numeric vector [1, 0, 2, 1, 0, 1]. That means that there is one *a*, zero *b's*, two *c's*, one *x*, zero *y's* and one *z*. Once this preprocessing step has been completed, a set of numeric vectors is generated that represents the entries in the CVE list. Each dimension in the vector represents a word that may potentially cluster ten or more entries in the CVE (remember that the occurrence threshold for removing a word is set to 10). The set of significant words thus provides a way of defining a numeric vector that can be used across all the entries in the CVE list. In this way, the natural language descriptions in the CVE list are transformed into an *i*-dimensional training vector that is ready to be used by the SOM.

The most important field in the CVE list is the "Description" field, which contains the standard names given to the publicly known information security vulnerabilities in the form of natural English language. For example, "*Buffer overflow in NFS mountd gives root access to remote attackers, mostly in Linux systems*" is the description given to the Vulnerability Name CVE-1999-0002. Words such as "*in*" and "*to*" are common to many entries in the CVE and carry no specific meaning, thus these words can be removed to shorten the sentence. This will place more emphasis on important words such as "*Buffer*" and "*overflow*". The resultant string after data cleaning has been performed is thus "*Buffer overflow NFS root access remote attackers Linux*". By removing all the insignificant words, we have simplified the training process for the SOM and prevented clustering around those insignificant words. In this research, a Java program was developed to do the data preprocessing and the transformation of the CVE list. A text file is generated that contains the numeric vectors for all CVE entries. A set of 394 significant words were found, which implies that the dimension of the numeric vectors is 394 (one dimension for each significant word). The Significant Word Set is given in Appendix B.

Table 5.3 contains more examples of data preprocessing. These examples display various types of descriptions. The Pattern column contains the resultant 394-dimensional numeric vectors for a particular CVE entry. Once the data preprocessing is complete, the results are saved in a text file that is used as input by the data-clustering step.

*Table 5.3 Extract from CVE entries after Data Preprocessing*

| Key | Description | After Data Preprocessing | Pattern |
|---|---|---|---|
| CVE-1999-0043 | Command execution via shell metachars in INN daemon (innd) 1.5 using "newgroup" and "rmgroup" control messages, and others. | execution shell daemon | 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 |
| CVE-1999-0044 | fsdump command in IRIX allows local users to obtain root access by modifying sensitive files. | IRIX root access | 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 |
| CVE-1999-0047 | MIME conversion buffer overflow in sendmail versions 8.8.3 and 8.8.4. | MIME conversion buffer overflow sendmail | 0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0 |
| CVE-1999-0050 | Buffer overflow in HP-UX newgrp program | Buffer overflow HP-UX | 0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, |

| Key | Description | After Data Preprocessing | Pattern |
|---|---|---|---|
| | | | 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0 |
| CVE-1999-0067 | CGI phf program allows remote command execution through shell metacharacters. | CGI remote execution shell metacharacters | 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 |

### 5.3.1.3 Step 3: Data Clustering

Step 2 produced a text file containing all transformed numeric vector representations of the CVE entries. This file is then fed to the SOM tool to perform clustering. Due to limited commercial SOM tools being available on the market and problematic licensing issues, a SOM tool was developed by the researcher for the purposes of this research. It is a basic implementation of the SOM algorithm and forms part of the prototype – the Vulnerability Analyser (VA) – that will be discussed in detail in the next chapter. Data clustering performed by the SOM will group similar vulnerabilities into clusters. For example, all buffer overflow vulnerabilities will be grouped into one cluster.

### 5.3.1.4 Step 4: Inspecting and Labelling clusters

After the training has been completed, the SOM creates the specified output files. The clusters are not yet labelled with meaningful names and they only bear a uniquely generated cluster label to identify the different clusters. Thus, human intervention is needed to interpret the clusters and label them with a meaningful name. For example, the SOM will randomly assign a name to a generated cluster as CLUSTER_187. After inspecting the cluster, it is discovered that the vulnerabilities in this cluster are all related to buffer overflows. The cluster label is consequently changed to "Buffer Overflow". Details of outputs produced are discussed in the next chapter.

## 5.3.2 Phase 2: Classification

New vulnerabilities are found and published every day – thus it is important to ensure that the categorisation process is capable of handling all newly discovered vulnerabilities as they become available to the public. The classification depends on the results of the initial categorisation phase and, as opposed to this phase (which is a once-off process), the classification phase is an on-going process that is used to maintain the Categorised CVE database. The classification phase is shown in Figure 5.3.

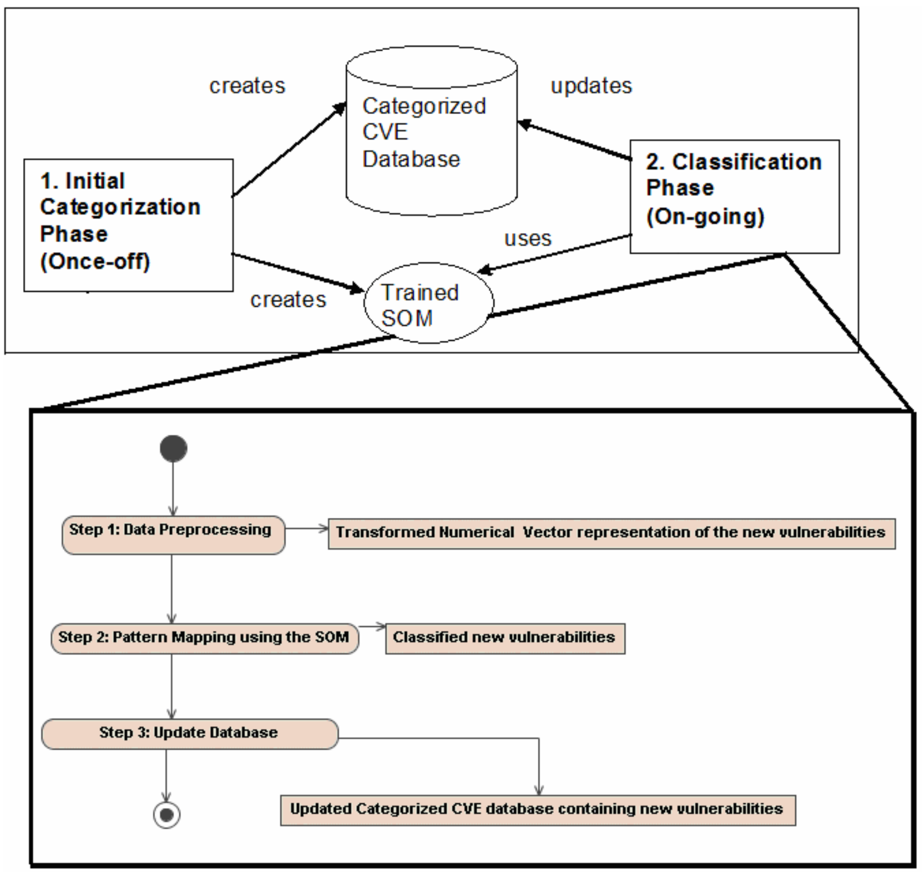

creates
Categorized CVE Database
updates
1. Initial Categorization Phase (Once-off)
2. Classification Phase (On-going)
uses
creates
Trained SOM
Step 1: Data Preprocessing
Transformed Numerical Vector representation of the new vulnerabilities
Step 2: Pattern Mapping using the SOM
Classified new vulnerabilities
Step 3: Update Database
Updated Categorized CVE database containing new vulnerabilities

***Figure 5.3 Classifying new vulnerabilities***

As new vulnerabilities are published by CVE, the following steps will classify them into correct categories:

1. Data preprocessing
2. Pattern mapping by the SOM
3. Updating the Categorised CVE database

New vulnerabilities are written in natural language as they are published by CVE; therefore the same data-preprocessing step is needed to transform them into a format that the SOM can use, as explained before. After the initial categorisation, the SOM is trained and will be able to classify new unseen vulnerabilities using its mapping phase. The numeric vectors produced by the data-preprocessing step are saved into a text file that has the same format as the patterns.txt file. The SOM classifies the numeric vectors by finding the best matching cluster for the numeric vector. Pattern mapping receives, as its input, a vector that has the same format as the input vectors. The values in the vector are used to find the neuron that best matches these values. Once the best matching neuron is found, that neuron's cluster is reported as the cluster for the vector. Similar output files are produced by the SOM to classify new vulnerabilities. When all new vulnerabilities have been classified, they are saved to the Categorised CVE database with all their information such as CVE KEY, Description, References, etc. For example, the classification phase classifies a CVE candidate as shown in Table 5.4, the entry is mapped to the "Denial of Service" cluster and the result is saved in the Categorised CVE database.

***Table 5.4 Example of classifying a CVE Candidate entry in the Classification Phase***

| Categorised CVE | | | |
|---|---|---|---|
| **CVE Key** | **Description** | **Pattern** | **Cluster** |
| CAN-1999-0001 | Denial of service in BSD-derived TCP/IP implementations, as described in CERT CA-98-13. | 0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,<br>0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0<br>,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 | Denial of Service |

## 5.4 CONCLUSION

This chapter has attempted to provide solutions to the research problem: How can the disparity in current vulnerability scanners be solved? The categorisation process was explained, as well as the CVE list that addresses the naming disparity in current vulnerability scanners. The categorisation process proposed by this research aims to take standardisation further by generating standardised vulnerability categories with a view to addressing the categorisation disparity in current vulnerability scanners. The categorisation process contains two phases: Initial Categorisation and Classification.

The Initial Categorisation phase involves constructing a standard set of vulnerability categories from the CVE list using a SOM. It consists of four steps: Acquire data source; Date preprocessing; Data clustering using a SOM, and Inspection and labelling. The first step, Acquire data source, involves downloading the CVE list from its website and importing it to a database for further use as the data source for the research. Through data preprocessing, the data in the data source is cleaned and transformed into a format that can be used by the SOM tool. Data clustering uses a SOM tool that was developed by the author to perform clustering. After the results are produced by the SOM tool, the last step involves inspecting the clusters formed and labelling them accordingly. The final results of the clustering are then saved and outputted to a database.

Chapter 5 also shows how the categorisation process is scalable and able to handle newly-found vulnerabilities as they are published. The Classification Phase is an ongoing process that continuously updates the Categorised CVE database with new vulnerabilities and their category. This is important in order to keep the categorisation results up to date with the market. The Classification Phase consists of three steps, namely Data preprocessing, Pattern Mapping by SOM, and Updating of the Categorised CVE database.

This chapter lays the foundation for building a prototype to automate vulnerability assessment. The next chapter presents and evaluates the prototype, Vulnerability Analyser (VA), which was developed for this research to demonstrate the categorisation process. The vulnerability categories created by the prototype are discussed and critically evaluated. **Chapter 6** discusses previous work on categorisation and vulnerability scanner assessment in brief, but refers to the benefits of standardising vulnerability categories in detail.

# CHAPTER 6

# 6 VULNERABILITY ANALYSER – PROTOTYPE AND EMPIRICAL ANALYSIS

## 6.1 *INTRODUCTION*

In the previous chapter, the categorisation process, including its two phases, was explained in detail. This proposed methodology aimed to standardise vulnerability categories so as to address some of the shortfalls of current vulnerability scanners.

Chapter 6 is organised as follows: the first section (Section 6.2) discusses in detail the prototype developed for the categorisation process. Section 6.3 introduces and explains the evaluation criteria by means of which the vulnerability categories generated are critically evaluated. Section 6.4 discusses the benefits of having a standard set of vulnerability categories, and the chapter is concluded in Section 6.5.

## 6.2 *EXPERIMENTAL RESULTS*

The following subsections discuss the configuration of the prototype, the experimental procedure, and experimental results of the two phases of the categorisation process respectively.

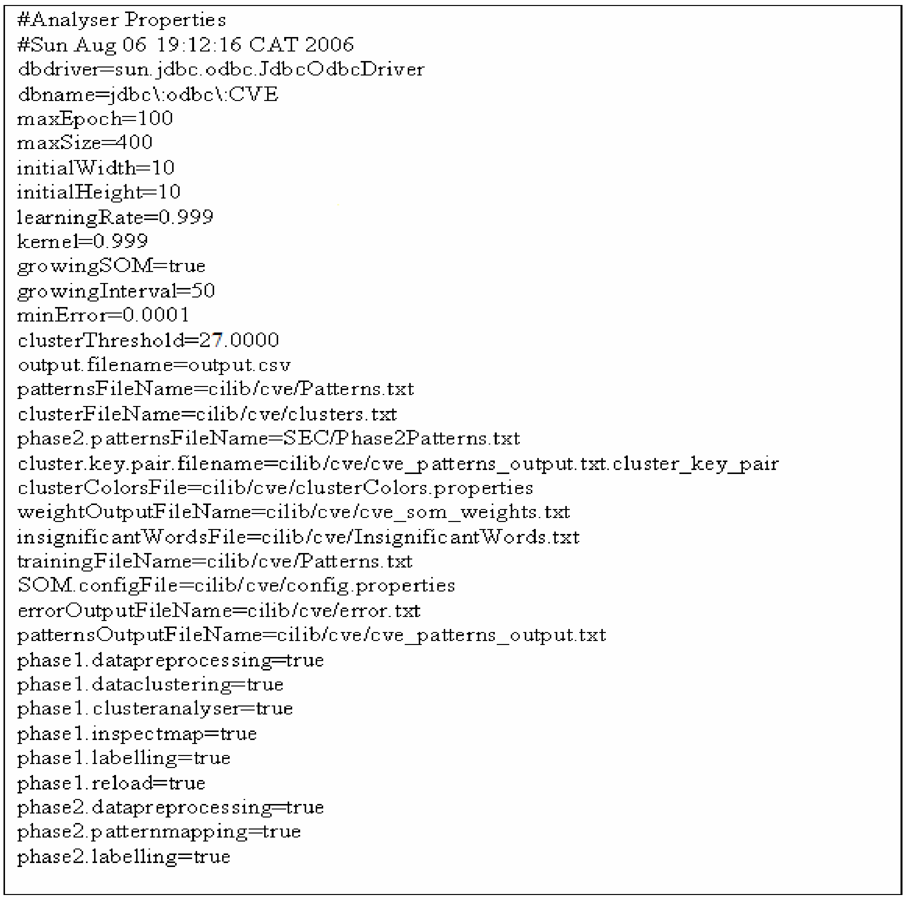
### 6.2.1 Prototype – Vulnerability Analyser (VA)

The prototype, known as the Vulnerability Analyser (VA), is developed to implement the entire two-phased categorisation process. Essentially, it is a SOM implementation with other functionalities that simplifies the cluster analysis and labelling process.

When running the prototype, the user has to provide all the input parameters that the VA requires. This set of input parameters is explained in the subsection that follows.

### 6.2.1.1 VA’s input parameters

All the input parameters required are saved into a properties file. The purpose of the properties file is to customise the SOM for this research problem space. Figure 6.1 shows such a properties file. Each parameter in the file is explained below.



```
#Analyser Properties
#Sun Aug 06 19:12:16 CAT 2006
dbdriver=sun.jdbc.odbc.JdbcOdbcDriver
dbname=jdbc\:odbc\:CVE
maxEpoch=100
maxSize=400
initialWidth=10
initialHeight=10
learningRate=0.999
kernel=0.999
growingSOM=true
growingInterval=50
minError=0.0001
clusterThreshold=27.0000
output.filename=output.csv
patternsFileName=cilib/cve/Patterns.txt
clusterFileName=cilib/cve/clusters.txt
phase2.patternsFileName=SEC/Phase2Patterns.txt
cluster.key.pair.filename=cilib/cve/cve_patterns_output.txt.cluster_key_pair
clusterColorsFile=cilib/cve/clusterColors.properties
weightOutputFileName=cilib/cve/cve_som_weights.txt
insignificantWordsFile=cilib/cve/InsignificantWords.txt
trainingFileName=cilib/cve/Patterns.txt
SOM.configFile=cilib/cve/config.properties
errorOutputFileName=cilib/cve/error.txt
patternsOutputFileName=cilib/cve/cve_patterns_output.txt
phase1.datapreprocessing=true
phase1.dataclustering=true
phase1.clusteranalyser=true
phase1.inspectmap=true
phase1.labelling=true
phase1.reload=true
phase2.datapreprocessing=true
phase2.patternmapping=true
phase2.labelling=true
```

***Figure 6.1 Properties file used for the SOM tool***

The **dbdriver** and **dbname** specify the CVE database properties. The **maxEpoch** defines the number of epochs for the training process, where each epoch is a single training cycle. Once this maximum number is reached, the SOM training process is terminated. For the purposes of this research, the SOM training process was run a number of times with various values for the maxEpoch. The experimental values used in this research for maxEpoch are 10, 50 and 100.

The **learningRate** value defines the initial learning rate, which should be less than 1.0 [ENGE 02]. The learning rate determines how the SOM is updated after each epoch. This learning rate decays during training, and the rate of decay is determined by the following formula:

$$\textbf{New learningRate} = 1.0 - \left(1.0/e^{((1.0 + \text{old learningRate})/(t + 1))}\right) \quad \text{[ENGE 02]}$$

where *t* represents the epoch number. The learning rate is decreased as the weights need to be adjusted by a smaller amount in order for the SOM to converge to a solution. The SOM converges to a solution as the quantisation error decreases.

The **kernel** is the diameter of the Gaussian function. It defines a radius that determines how neighbouring units around a winning neuron are updated. If the kernel is large, the SOM updates more surrounding neighbouring units, whereas for a small kernel, fewer neighbouring units are updated. Similar to the learning rate, the kernel also decays during training. Thus it is set to be 0.999 initially, and the rate of decay is determined by the formula:

$$\textbf{New kernel} = 1.0 - \left(1.0/e^{((1.0 + \text{old kernel})/(t + 1))}\right) \quad \text{[ENGE 02]}$$

where *t* represents the epoch number.

The minError is another termination criterion for training the SOM. The error is the SOM’s quantisation/training error, which is calculated after each epoch. This value describes the accuracy or “how well” the SOM has trained. Smaller values indicate a better map as training vectors match a unit on the SOM with high accuracy. The minError was arbitrarily chosen as 0.0001 for this research.

During training, the Growing SOM adds a number of neurons as needed, resulting in an architecture that is best suitable for the input data set. Refer again to Section 4.6 for a detailed explanation of the Growing SOM. For this research the Growing SOM is used, therefore the **growingSOM** indicator is set to true. The **growingInterval** defines the epoch interval when the SOM is to grow. During the growing process, additional neuron units are added to the SOM to reduce the quantisation error. The **maxSize** defines the maximum map size the SOM can reach. A very large map size is not necessarily good, since such an attempt will result in many units not having any patterns matched to them. On the other hand, a small map size does not distribute data properly, which results in maps that are too difficult to analyse since the input patterns are compressed too much and not many conclusions will be drawn from such a map. An appropriate size has to be chosen based on the size of the training patterns provided. For this research, the **maxSize** of 400 neurons (a 20 x 20 map) and the **growingInterval** of every 50th epoch seem to show the best results.

The initial width and height of the Growing SOM are usually small. These values are specified by the properties **initialWidth** and **initialHeight** in the properties file, namely 10 and 10 respectively to avoid a too large map from the start of the training.

In the clustering process, all neurons are initially identified as clusters. Clusters that are neighbours and have a distance of less than a threshold between each other are merged together. The threshold is defined as the **clusterThreshold**. The cluster threshold controls the number of clusters that can be formed. A high cluster threshold generates too few clusters where each cluster is large. Large clusters group too many neurons (or patterns) that can be further split into smaller clusters. A low cluster threshold generates too many small clusters. Small clusters group too few patterns and thus do not offer any useful information. For this research, it was empirically determined that the threshold value of 27.00 provides the best number of clusters (however, this value can be adjusted to change the number clusters that are formed).

A set of parameters is used to help the VA to locate necessary input and output files. For example, the Training File Name specifies the full path to the input data set – Patterns.txt. The training patterns are shuffled before each epoch. This prevents the SOM from memorising the pattern orders during training and improves the learning quality of the SOM. This set of parameters specifies the location of all the necessary files requested by the SOM.

The rest of the parameters are used to save the application status of the VA, which allows the users to terminate the application at any time of execution. All application data and status are persisted to a file, so the next time when they use the application, they can continue with the process instead of having to restart from the beginning.

After training, the final width and height of the SOM map are appended to the properties file. The next subsection explains the screen layouts of the VA.

#### 6.2.1.2 VA's Graphical User Interface

The VA screen for Phase 1, shown in Figure 6.2, contains the various steps in the Initial Categorisation Process. The Data Acquisition step should already be completed before running the

VA, that is, the CVE list should already be loaded into the tables in the database. Selecting the **Data Preprocessing** button will perform the data-preprocessing step, and selecting the **Data-clustering** button will perform the clustering step. The **Inspect Map**, **Labelling**, **Cluster Analyser** and **Reload** buttons are for the inspecting and labelling clusters step. These four processes can be executed repetitively to finalise the clustering results. **Inspect Map** shows the maps that have been generated by the VA, and a colour picker allows the user to change the names as well as the colours for clusters. Three maps are generated in this step; they are the Frequency Map, U-Matrix Map and the Cluster Map, as shown in Figure 6.3. The Frequency map, U-Matrix map, cluster map and colour picker are discussed later in this section. **Labelling cluster** updates the CVE entries in the Categorised CVE table to include the category names. **Cluster Analyser** displays the centroid of the clusters and the word count, so the user can easily find words that a cluster represents, as shown in Figure 6.4. **Reload** allows the user to change the cluster threshold for the SOM and regenerate the clusters.

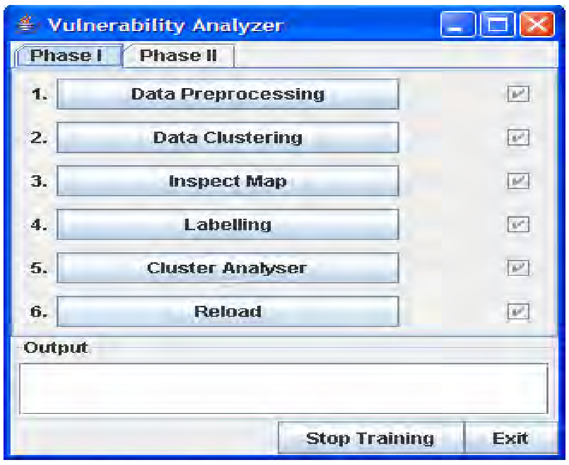


*Figure 6.2 Initial Categorisation Phase Screen*

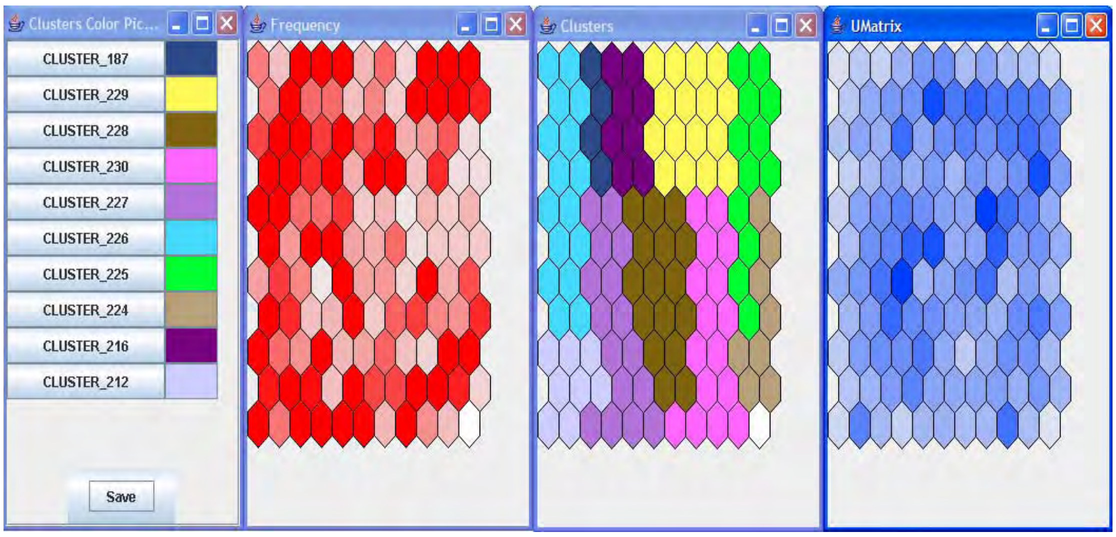
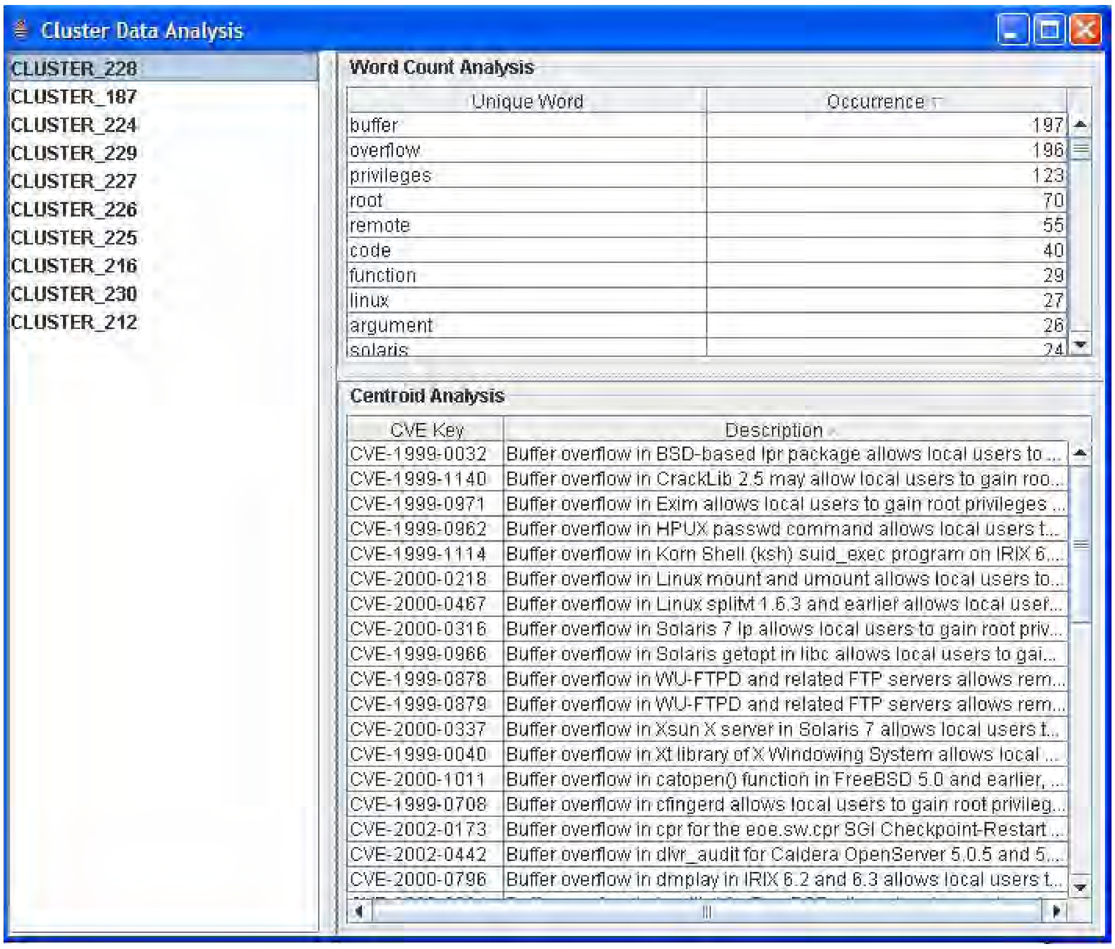


*Figure 6.3 Sample Maps generated by Inspect Map*



| Unique Word | Occurrence |
|---|---|
| buffer | 197 |
| overflow | 196 |
| privileges | 123 |
| root | 70 |
| remote | 55 |
| code | 40 |
| function | 29 |
| linux | 27 |
| argument | 26 |
| solaris | 24 |

| CVE Key | Description |
|---|---|
| CVE-1999-0032 | Buffer overflow in BSD-based lpr package allows local users to ... |
| CVE-1999-1140 | Buffer overflow in CrackLib 2.5 may allow local users to gain roo... |
| CVE-1999-0971 | Buffer overflow in Exim allows local users to gain root privileges ... |
| CVE-1999-0962 | Buffer overflow in HPUX passwd command allows local users t... |
| CVE-1999-1114 | Buffer overflow in Korn Shell (ksh) suid_exec program on IRIX 6,... |
| CVE-2000-0218 | Buffer overflow in Linux mount and umount allows local users to... |
| CVE-2000-0467 | Buffer overflow in Linux splitvt 1.6.3 and earlier allows local user... |
| CVE-2000-0316 | Buffer overflow in Solaris 7 lp allows local users to gain root priv... |
| CVE-1999-0966 | Buffer overflow in Solaris getopt in libc allows local users to gai... |
| CVE-1999-0878 | Buffer overflow in WU-FTPD and related FTP servers allows rem... |
| CVE-1999-0879 | Buffer overflow in WU-FTPD and related FTP servers allows rem... |
| CVE-2000-0337 | Buffer overflow in Xsun X server in Solaris 7 allows local users t... |
| CVE-1999-0040 | Buffer overflow in Xt library of X Windowing System allows local ... |
| CVE-2000-1011 | Buffer overflow in catopen() function in FreeBSD 5.0 and earlier, ... |
| CVE-1999-0708 | Buffer overflow in cfingerd allows local users to gain root privileg... |
| CVE-2002-0173 | Buffer overflow in cpr for the eoe.sw.cpr SGI Checkpoint-Restart ... |
| CVE-2002-0442 | Buffer overflow in dlvr_audit for Caldera OpenServer 5.0.5 and 5... |
| CVE-2000-0796 | Buffer overflow in dmplay in IRIX 6.2 and 6.3 allows local users t... |

*Figure 6.4 Sample Screens generated by Cluster Analyser*

The process of converting the CVE descriptions into numerical vectors is regarded as a preprocessing step, and thus it is executed only once. The SOM is trained on these numerical vectors and the resulting clusters are evaluated at 10, 50 and 100 maximum epochs. The cluster threshold is adjusted to be the average distance between all neurons in the SOM. The generated clusters are inspected by determining the centroid of the cluster, and then finding which patterns map to the centroid of the cluster. The patterns that map to the centroid of the cluster best describe the cluster and can be used to label the cluster. If it is not possible to determine cluster labels, then either the SOM needs to train for a larger number of epochs, or the cluster threshold is incorrect.

The **Clusters Colour Picker** allows the user to change the colour of the clusters as well as the cluster name by double clicking on either the colour display or on the cluster label in the picker frame. Thus, after the inspection, a proper cluster name can be assigned to the clusters using **the Cluster Colour Picker**. The **Label cluster** step is responsible for updating the cluster labels for the Categorised CVE table. Figure 6.3 shows an example of a Clusters Colour Picker.

After the Initial Categorisation Phase, the Classification Phase for vulnerabilities that are not part of the CVE can proceed, as shown in Figure 6.5. It contains three steps that are very similar to the initial categorisation phase. The **Data-Preprocessing** step performs data preprocessing on the new vulnerabilities, **Pattern Mapping** uses the trained SOM from the first phase to map new vulnerabilities to their closest cluster, and **Labelling** saves the results to the Categorised CVE table.

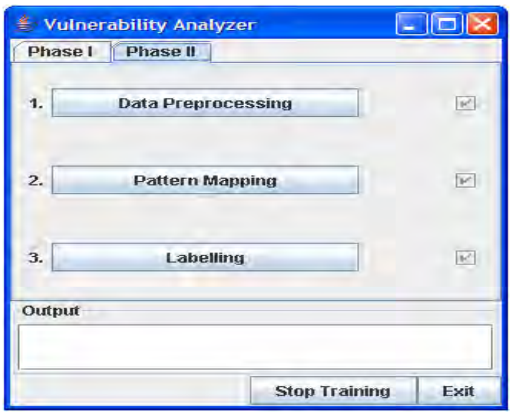
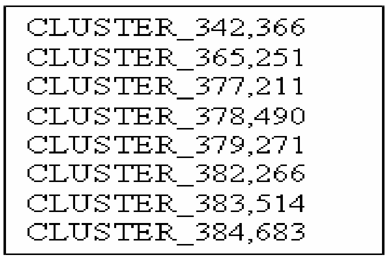


***Figure 6.5 The Classification Phase screen***

The results of the VA are discussed in the next subsection.

### 6.2.1.3 VA's Outputs

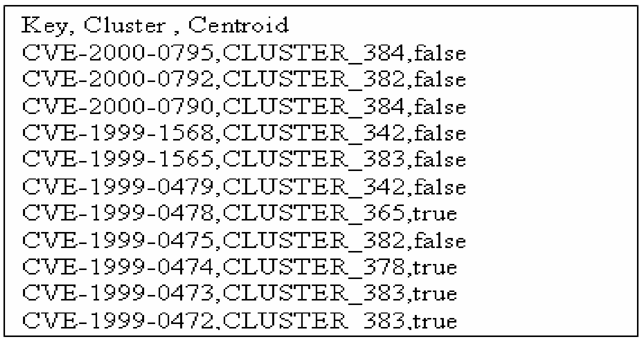
The prototype generates several outputs. Figure 6.6 shows an example of the clusters.txt file. The output contains cluster labels (at the moment, these labels are system generated) and the number of CVE entries for each cluster.



```
CLUSTER_342,366
CLUSTER_365,251
CLUSTER_377,211
CLUSTER_378,490
CLUSTER_379,271
CLUSTER_382,266
CLUSTER_383,514
CLUSTER_384,683
```

***Figure 6.6 The clusters.txt file produced by VA***

Figure 6.7 shows an extract from the cluster_key_pair file. Each CVE entry is assigned with a cluster. The centroid of a cluster is a neuron unit that is the centre of a cluster. Patterns that are mapped to the centroid can be seen as the cluster representatives. In this figure, the Key column contains the CVE key of the data entry, the Cluster column shows the name of the cluster that this entry belongs too, and the Centroid column indicates whether this entry is the centroid of the cluster or not.



```
Key, Cluster , Centroid
CVE-2000-0795,CLUSTER_384,false
CVE-2000-0792,CLUSTER_382,false
CVE-2000-0790,CLUSTER_384,false
CVE-1999-1568,CLUSTER_342,false
CVE-1999-1565,CLUSTER_383,false
CVE-1999-0479,CLUSTER_342,false
CVE-1999-0478,CLUSTER_365,true
CVE-1999-0475,CLUSTER_382,false
CVE-1999-0474,CLUSTER_378,true
CVE-1999-0473,CLUSTER_383,true
CVE-1999-0472,CLUSTER_383,true
```

***Figure 6.7 Extract from the cluster_key_pair file produced by VA***

Various SOM maps are also outputted by VA, thus aiding the inspection process. The U-Matrix, Frequency Map and the Cluster Map are all particularly useful and are explained below.

The Cluster Map reveals clusters of similar data. Different colours are assigned to different clusters. An example of a Cluster Map is shown in Figure 6.3, where each colour represents a cluster and the white areas represent neuron units that do not map to any training pattern.

The Frequency map reveals the number of input vectors that mapped to each unit on the SOM map. A dark red colour indicates a large frequency and a lighter (grey to white) colour indicates a lower frequency of input vectors for a unit. White units indicate 0 frequency. The sole purpose of using this map is for the user to investigate how the input vectors have been distributed by SOM. A consistent colour throughout the map indicates an even distribution, which means that the data has spread throughout the map, making it easier for analysis and exploration. An example of a Frequency Map is shown in Figure 6.3.

An example of the U-Matrix Map is also shown in Figure 6.3. The U-Matrix map indicates the distance from a neuron to its surrounding units. A lighter blue to grey colour indicates a large distance, while darker blue colours indicate smaller distances. From this figure it is clear that the neurons are closer to each other around the centre and in the top right corner. This map can be used in conjunction with the Cluster Map to help define cluster boundaries. Darker areas of the map are separated by lighter areas as the lighter areas define the cluster boundaries as shown in the Cluster Map.

The results in the ***cluster_key_pair*** file need to be transferred into the database for further use, because it is easier to work with database tables than with plain text files. A categorised CVE table is created in the database that contains the following fields: **CVEKey**, **Description**, **Pattern**, **Cluster** and **Centroid**. **CVEKey** is the primary key field that contains the CVE keys; **Description** contains the original CVE descriptions, and **Pattern** contains the numeric vectors that are fed to the SOM tool as inputs. **Labelling cluster** updates the **Cluster** column of the categorised CVE table with the cluster labels in the ***cluster_key_pair*** file. The **Centroid** indicator shows whether the pattern has been mapped to the centre of a cluster or not.

### 6.2.2 Results of the Initial Categorisation Phase

The results of the SOM are presented in this section for the 100th epoch. The centroids of the various clusters are inspected, and labels are determined if possible.

**Results after 100 epochs**

As discussed in the previous section, for 100 epochs, the cluster threshold was determined to be 27.00. The number of clusters generated with a cluster threshold of 27.00 was 7. Figure 6.8 shows the various maps generated by VA. The average frequency of the SOM neuron units is 25.223, which means that on average 25 input patterns are mapped to a single neuron unit in the SOM. From the Frequency map it is clear that there is a fair distribution of the patterns across the map. Cluster Map shows the cluster formations and U-Matrix Map shows the distance between the neurons. Refer to the previous section for more detail on the maps.

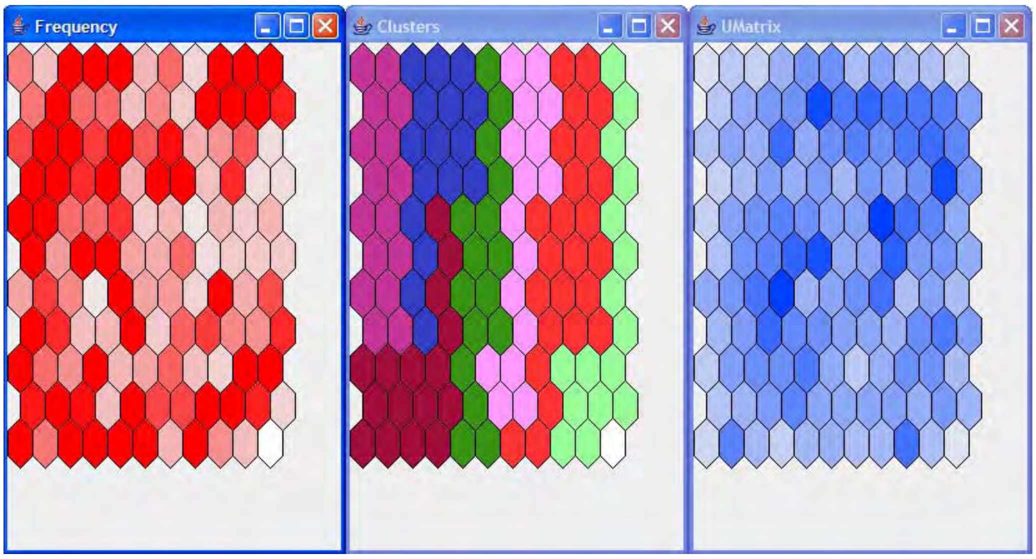


*Figure 6.8 The Frequency, Clusters and U-Matrix Maps generated after 100 epochs*

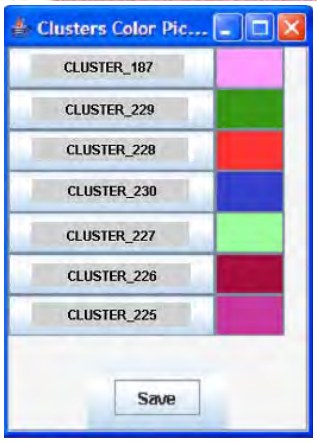


***Figure 6.9 The Colour Picker generated after 100 epochs***

Figure 6.9 shows the system-generated cluster labels. Cluster Analyser assists with investigating and understanding the generated clusters. Cluster labels are only assigned after the inspection of the clusters based on the word occurrences and string patterns in each cluster. Table 6.1 shows a few CVE entries that have mapped to the centroids of the respective clusters and the assigned cluster labels.

***Table 6.1 Extracts of Centroids for each generated cluster and assigned Cluster Labels***

CLUSTER_187: Buffer Overflow

| | |
|---|---|
| CVE-2000-0477 | Buffer overflow in Norton Antivirus for Exchange (NavExchange) allows remote attackers to cause a denial of service via a .zip file that contains long file names. |
| CVE-2000-0257 | Buffer overflow in the NetWare remote web administration utility allows remote attackers to cause a denial of service or execute commands via a long URL. |
| CVE-2001-0035 | Buffer overflow in the kdc_reply_cipher function in KTH Kerberos IV allows remote attackers to cause a denial of service and possibly execute arbitrary commands via a long authentication request. |
| CVE-2000-0943 | Buffer overflow in bftp daemon (bftpd) 1.0.11 allows remote attackers to cause a denial of service and possibly execute arbitrary commands via a long USER command. |
| CVE-2001-0233 | Buffer overflow in micq client 0.4.6 and earlier allows remote attackers to cause a denial of service, and possibly execute arbitrary commands, via a long Description field. |

CLUSTER_229: Privilege Escalation

| | |
|---|---|
| CVE-2000-0763 | xlockmore and xlockf do not properly cleanse user-injected format strings, which allows local users to gain root privileges via the -d option. |
| CVE-2003-0019 | uml_net in the kernel-utils package for Red Hat Linux 8.0 has incorrect setuid root privileges, which allows local users to modify network interfaces, e.g. by modifying ARP entries or placing interfaces into promiscuous mode. |
| CVE-1999-0420 | umapfs allows local users to gain root privileges by changing their uid through a malicious mount_umap program. |
| CVE-2002-0043 | sudo 1.6.0 through 1.6.3p7 does not properly clear the environment before calling the mail program, which could allow local users to gain root privileges by modifying environment variables and changing how the mail program is invoked. |

| | |
|---|---|
| CVE-2004-0186 | smbmnt in Samba 2.x and 3.x on Linux 2.6, when installed setuid, allows local users to gain root privileges by mounting a Samba share that contains a setuid root program, whose setuid attributes are not cleared when the share is mounted. |

CLUSTER_228 Software Vulnerabilities

| | |
|---|---|
| CVE-2000-0707 | PCCS MySQLDatabase Admin Tool Manager 1.2.4 and earlier installs the file dbconnect.inc within the web root, which allows remote attackers to obtain sensitive information such as the administrative password. |
| CVE-2001-1372 | Oracle 9i Application Server 1.0.2 allows remote attackers to obtain the physical path of a file under the server root via a request for a non-existent .JSP file, which leaks the pathname in an error message. |
| CVE-2002-0898 | Opera 6.0.1 and 6.0.2 allows a remote web site to upload arbitrary files from the client system, without prompting the client, via an input type=file tag whose value contains a newline. |
| CVE-2002-0209 | Nortel Alteon ACEdirector WebOS 9.0, with the Server Load Balancing (SLB) and Cookie-Based Persistence features enabled, allows remote attackers to determine the real IP address of a web server with a half-closed session, which causes ACEdirector to send packets from the server without changing the address to the virtual IP address. |
| CVE-2000-0406 | Netscape Communicator before version 4.73 and Navigator 4.07 do not properly validate SSL certificates, which allows remote attackers to steal information by redirecting traffic from a legitimate web server to their own malicious server, aka the "Acros-Suencksen SSL" vulnerability. |

CLUSTER_230: Denial of Service

| | |
|---|---|
| CVE-2001-0018 | Windows 2000 domain controller in Windows 2000 Server, Advanced Server, or Datacenter Server allows remote attackers to cause a denial of service via a flood of malformed service requests. |
| CVE-2000-0593 | WinProxy 2.0 and 2.0.1 allows remote attackers to cause a denial of service by sending an HTTP GET request without listing an HTTP version number. |
| CVE-2000-0738 | WebShield SMTP 4.5 allows remote attackers to cause a denial of service by sending e-mail with a From: address that has a . (period) at the end, which causes WebShield to continuously send itself copies of the e-mail. |
| CVE-2000-1182 | WatchGuard Firebox II allows remote attackers to cause a denial of service by flooding the Firebox with a large number of FTP or SMTP requests, which disables proxy handling. |
| CVE-2000-0644 | WFTPD and WFTPD Pro 2.41 allows remote attackers to cause a denial of service by executing a STAT command while the LIST command is still executing. |

CLUSTER_227: Scripting Metacharacters

| | |
|---|---|
| CVE-2002-0682 | Cross-site scripting vulnerability in Apache Tomcat 4.0.3 allows remote attackers to execute script as other web users via script in a URL with the /servlet/ mapping, which does not filter the script when an exception is thrown by the servlet. |
| CVE-2002-1195 | Cross-site scripting vulnerability (XSS) in the PHP interface for ht://Check 1.1 allows remote web servers to insert arbitrary HTML, including script, via a web page. |
| CVE-2003-0002 | Cross-site scripting vulnerability (XSS) in ManualLogin.asp script for Microsoft Content Management Server (MCMS) 2001 allows remote attackers to execute arbitrary script via the REASONTXT parameter. |
| CVE-2002-1307 | Cross-site scripting vulnerability (XSS) in MHonArc 2.5.12 and earlier allows remote attackers to insert script or HTML via an email message with the script in a |

| | |
|---|---|
| | MIME header name. |
| CVE-2002-0840 | Cross-site scripting (XSS) vulnerability in the default error page of Apache 2.0 before 2.0.43, and 1.3.x up to 1.3.26, when UseCanonicalName is "Off" and support for wildcard DNS is present, allows remote attackers to execute script as other web page visitors via the Host: header, a different vulnerability than CAN-2002-1157. |

CLUSTER_226: Data Corruption

| | |
|---|---|
| CVE-2000-0076 | nviboot boot script in the Debian nvi package allows local users to delete files via malformed entries in vi.recover. |
| CVE-1999-0300 | nis_cachemgr for Solaris NIS+ allows attackers to add malicious NIS+ servers. |
| CVE-2002-0355 | netstat in SGI IRIX before 6.5.12 allows local users to determine the existence of files on the system, even if the users do not have the appropriate permissions. |
| CVE-2003-0924 | netpbm 9.25 and earlier does not properly create temporary files, which allows local users to overwrite arbitrary files. |
| CVE-2002-1518 | mv in IRIX 6.5 creates a directory with world-writable permissions while moving a directory, which could allow local users to modify files and directories. |

CLUSTER_225: Information Gathering

| | |
|---|---|
| CVE-2000-0728 | xpdf PDF viewer client earlier than 0.91 allows local users to overwrite arbitrary files via a symlink attack. |
| CVE-2002-0213 | xkas in Xinet K-AShare 0.011.01 for IRIX allows local users to read arbitrary files via a symlink attack on the VOLICON file, which copied to the .HSicon file in a shared directory. |
| CVE-2001-0887 | xSANE 0.81 and earlier allows local users to modify files of other xSANE users via a symlink attack on temporary files. |
| CVE-2001-0222 | webmin 0.84 and earlier allows local users to overwrite and create arbitrary files via a symlink attack. |
| CVE-2001-0143 | vpop3d program in linuxconf 1.23r and earlier allows local users to overwrite arbitrary files via a symlink attack. |

The next subsection discusses the results obtained from the second phase of the categorisation process.

### 6.2.3 Results of the Classification Phase

The classification phase essentially tests the generalisation ability of the SOM. In the case of unknown vulnerabilities, the SOM will classify these new vulnerabilities into a category that is best suitable for the new vulnerability. To test the classification, a group of entries is selected from the CVE with status “Candidate” (Keys prefaced by “CAN-“, which is never used in training) as the data source. A text file is generated after the data-preprocessing step, which contains the numerical vector representation of the new vulnerabilities.

Twenty CAN entries are processed by the VA in the classification phase, and the results of their classifications are show in Table 6.2 below. It shows for each CAN entry the specific cluster that it maps to, as well as whether it has been mapped to the cluster centroids neuron or not.

***Table 6.2: Results of the Classification Phase***

| CVEKey | Description | Cluster | Centroid |
|---|---|---|---|
| CAN-1999-0001 | Denial of service in BSD-derived TCP/IP implementations, as described in CERT CA-98-13. | Denial of Service | false |
| CAN-1999-0004 | MIME buffer overflow in email clients, e.g. Solaris mailtool and Outlook. | Software Vulnerabilities | false |
| CAN-1999-0015 | Teardrop IP denial of service. | Denial of Service | false |
| CAN-1999-0030 | Root privileges via buffer overflow in xlock command on SGI IRIX systems. | Privilege Escalation | false |
| CAN-1999-0033 | Command execution in Sun systems via buffer overflow in the at program. | Software Vulnerabilities | false |
| CAN-1999-0061 | File creation and deletion, and remote execution, in the BSD line printer daemon (lpd). | Data Corruption | false |
| CAN-1999-0078 | pcnfsd (aka rpc.pcnfsd) allows local users to change file permissions, or execute arbitrary commands through arguments in the RPC call. | Data Corruption | false |
| CAN-1999-0104 | A later variation on the Teardrop IP denial of service attack, a.k.a. Teardrop-2. | Denial of Service | false |
| CAN-1999-0105 | Finger allows recursive searches by using a long string of @ symbols. | Data Corruption | true |
| CAN-1999-0106 | Finger redirection allows finger bombs. | Data Corruption | true |
| CAN-1999-0107 | Buffer overflow in Apache 1.2.5 and earlier allows a remote attacker to cause a denial of service with a large number of GET requests containing a large number of / characters. | Buffer Overflow | true |
| CAN-1999-0114 | Local users can execute commands as other users, and read other users' files, through the filter command in the Elm elm-2.4 mail package using a symlink attack. | Information Gathering | true |
| CAN-1999-0119 | Windows NT 4.0 beta allows users to read and delete shares. | Software Vulnerabilities | false |
| CAN-1999-0123 | Race condition in Linux mailx command allows local users to read user files. | Data Corruption | false |
| CAN-1999-0140 | Denial of service in RAS/PPTP on NT systems. | Denial of Service | false |
| CAN-1999-0144 | Denial of service in Qmail by specifying a large number of recipients with the RCPT command. | Denial of Service | false |
| CAN-1999-0154 | IIS 2.0 and 3.0 allow remote attackers to read the source code for ASP pages by appending a . (dot) to the end of the URL. | Software Vulnerabilities | false |
| CAN-1999-0169 | NFS allows attackers to read and write any file on the system by specifying a false UID. | Data Corruption | false |
| CAN-1999-0197 | Finger 0@host on some systems may print information on some user accounts. | Information Gathering | false |
| CAN-1999-0200 | Windows NT FTP server (WFTP) with the guest account enabled without a password allows an attacker to log into the FTP server using any username and password. | Software Vulnerabilities | false |

## 6.3 EVALUATION

To evaluate the results of the categorisation process in this research, a set of evaluation criteria are defined by the researcher:

- Similar vulnerabilities are clustered together.
- Mutually exclusive
- Exhaustive
- Unambiguous
- Repeatable
- Accepted
- Scalable
- Useful

Each of these criteria is now discussed in order to determine whether the categorisation process is reputable.

**Similar vulnerabilities have to be clustered together**. For example, from the result produced by the prototype, all vulnerabilities arising from Buffer Overflow are grouped together based on their common characteristics. Thus, the results obtained from the categorisation phase have shown that this criterion is indeed met by the prototype.

The categories generated need to be **mutually exclusive**, which means that vulnerabilities can receive one and only one category. Mutually exclusive categories eliminate confusion among categories. Although overlaps in categories may occur, vulnerabilities in one category are not required to be distinct from other vulnerabilities. For example, since a vulnerability involving a buffer overflow may lead to privilege escalations, it should also be grouped with the Privilege Escalation category. As a result, vulnerabilities may fall into multiple categories. Because a vulnerability can rarely be characterised in exactly one way, a realistic classification scheme must take the multiple characteristics of vulnerabilities into account. Although vulnerabilities can theoretically fall into multiple categories, they should always be classified into only one category to meet this criterion. The categorisation process will, under all circumstances, assign only one category to each vulnerability.

Categories will also need to be **exhaustive**. They should include all possibilities. That is, during the categorisation process, a category should always be found for a vulnerability. The results of the prototype shown that every vulnerability has been classified.

Categories need to be **clear and precise** so that classification is certain, regardless of who is categorising. In this research, the categorisation was done by the VA which uses a SOM (unsupervised learning data-clustering technique), therefore it is not biased by any person. Since it is unsupervised learning, it also does not need much prior knowledge from the user. Thus, the results produced do not depend on any specific person, and are unambiguous.

Categories should be **repeatable** – repeated applications of categorisation should result in the same result set, regardless of who is categorising. Multiple tests were done in this research and the results generated were found to be repeatable.

**Categories generated should be accepted.** The categories should be logical and intuitive so that they could become generally approved. The categories developed were based upon a logical process that was intended to be intuitive. This categorisation was designed to be widely accepted since it is based itself on CVE.

**Categories generated should be scalable**. The categories are scalable; new vulnerabilities can be added to each category as they get discovered. The classification phase of the categorisation process ensures scalability in that this categorisation can handle on-going classification of unknown vulnerabilities.

**Categories generated should be useful**. The final characteristic of a satisfactory taxonomy is that it can be used to gain insight into the field of inquiry. By going through the categorisation process defined in this research, people can gain insight into vulnerabilities through inspecting the SOM maps and analysing the clusters generated with Cluster Analyser, and they can assign meaningful names to the generated clusters.

The next section discusses how these standardised vulnerability categories can be useful to address some of the problems with current vulnerability scanners. In Chapter 7, an extension to the prototype is developed to further demonstrate the usefulness of these categories.

## *6.4 BENEFITS OF STANDARDISING VULNERABILITY CATEGORIES*

The main benefit of having a standard set of vulnerability categories is to enable the users to assess and compare various vulnerability scanners in order to determine their capabilities and pitfalls.

The benefits of standardising vulnerability categories are the following: it facilitates vulnerability scanner assessment and comparison, improves interoperability, produces summarised reports, and simplifies Vulnerability Databases' responsibilities. Each of the benefits is explained in the subsections below.

### 6.4.1 VS Assessment and Comparisons

Having a standard set of vulnerability categories is useful when conducting quantitative comparisons of various vulnerability scanners. It simplifies the investigation for vulnerability scanners by determining the number of vulnerability categories, as well as the number of vulnerabilities in each category that they can detect. This allows for the strengths and weaknesses of a vulnerability scanner to be identified. It also facilitates the comparison between various vulnerability scanners, which aids in making informed decisions when the best suitable scanner for an enterprise in terms of the enterprise's needs and priorities is to be elected.

### 6.4.2 Improved Interoperability

An array of vulnerability scanners can possibly be used to protect the enterprise's network and assets, where the weakness of one vulnerability scanner is the strength of another. The assessment of vulnerability scanners allows for the best suitable tools to be selected to provide coverage without reliance on a single vendor for a "suite" solution. This, in turn, enhances the communication and the security of organisations using vulnerability scanners.

### 6.4.3 Summarised Reports

Summarised reports can be produced, which provide a high-level overview of the vulnerabilities detected by the vulnerability categories. This allows the administrators to easily identify problem categories or shortfalls in the network. Such reports also allow effective analysis for determining macro-level vulnerability patterns. The efficiency and effectiveness of risk management can furthermore be improved since the reports produced are much shorter and intuitive, as opposed to lengthy scan reports. Administrators can therefore rectify the detected vulnerabilities more quickly.

### 6.4.4 Simplifying the Responsibilities of Vulnerability Databases

Similar to CVE, which addresses the naming disparity, the standard set of vulnerability categories is aimed at resolving the classification disparity among various vulnerability databases. This standard set of vulnerability categories is generated from the CVE list. Thus, all databases with references to the CVE are now able to link to the standard set of vulnerability categories. For those databases that do not have references to CVE, the classification phase of the categorisation process provides an option to get a reference to the standard set of vulnerability categories using the pattern mapping of SOM. Since all vulnerability databases are potentially capable of referencing the standard set of vulnerability categories, the requirement to generate vulnerability categories is completely removed, and the categorisation process is a simple mapping to the standard set of vulnerability categories. All vulnerabilities can now be categorised in a standard and consistent way.

## *6.5 CONCLUSION*

As shown in this chapter, a number of classification schemes have been developed by various parties for vulnerabilities, but there is no standard or collaboration between them. This research has proposed a methodology that attempts to address the classification disparity among vulnerability scanners and vulnerability databases. The prototype implementation of this methodology was discussed here. The results of the process are presented, as well as critically evaluated against the evaluation criteria. Seven vulnerability categories are produced, namely:

**Buffer Overflow**, **Denial of Service**, **Scripting Metacharacters**, **Privilege Escalation**, **Data Corruption**, **Information Gathering** and **Software Vulnerabilities**.

The main benefit of having such a standard set of vulnerability categories is to enable users to assess and compare vulnerability scanners so as to determine their capabilities and pitfalls. A standardised set of vulnerability categories will also allow vulnerability scanners to interoperate in a meaningful way to form an array of scanners that can be used to protect the enterprise's network and assets. Furthermore, summarised reports can be produced, which highlight the problem categories of the network. In other words, instead of presenting multiple vulnerabilities, a vulnerability category is presented with the exact number of vulnerabilities detected on the network that belongs to this category. This is shorter and less confusing, and can be interpreted more easily to (and by) a human.

The next chapter discusses an extension of the VA prototype developed for this research – the Vulnerability Product Assessor (VPA). It is aimed to prove the concept and benefits of having a standard set of categories for vulnerabilities.

# CHAPTER 7

# 7 VULNERABILITY PRODUCT ASSESSOR

## *7.1 INTRODUCTION*

The previous chapters discussed vulnerability scanners as well as the existing problems with this technology. One of the problems is that there is no clear, open and standard way of assessing vulnerability products. Network World Fusion has developed a set of criteria to compare vulnerability scanners, which includes Product or service; Product platform; Other platform; Net or host; CVE; Number of Vulnerabilities; Update links; How often; Autofix; Customising; Automation; Integration, and Pricing [NETW 04]. The only criteria that are important to this research are CVE and Number of Vulnerabilities. None of the other criteria, for example Pricing, are considered. CVE shows whether a specific vulnerability scanner has references to CVE. Number of Vulnerabilities shows the number of vulnerabilities in its vulnerability database. Just by looking at the size of the vulnerability database for a vulnerability scanner is not appropriate due to the disparity among current vulnerability scanners.

Chapter 7 introduces the Vulnerability Product Assessor (VPA) to demonstrate how the standard set of vulnerability categories generated by the VA in Chapter 6 can be used to resolve some of the problems faced by vulnerability products. Once the standard set of vulnerability categories has been constructed, it is possible to develop an application that automates the process of vulnerability tool assessment. In this chapter, a conceptual model for conducting automated vulnerability scanner assessment is proposed. The design of the model is discussed in detail and a prototype is developed to demonstrate the concept. Experimental results are also provided for the prototype.

The rest of the chapter is organised as follows: the conceptual design of the VPA is discussed in the next section. Section 7.3 discusses in detail the implementation of the VPA prototype developed for this research, and Section 7.4 concludes the chapter.

## 7.2 CONCEPTUAL DESIGN OF VULNERABILITY PRODUCT ASSESSOR (VPA)

The principal aim of the VPA is to assess various vulnerability products in a uniform way in order to automate vulnerability product assessment. The assessment is based on the content of the vulnerability databases contained in the products, in other words how many vulnerability categories a specific vulnerability scanner covers, or how comprehensive the vulnerability database is. The scan report produced by the vulnerability scanners can also be used to assess the products. For example, in the same computing environment, scanner A discovered *x* number of vulnerabilities more in category A than did scanner B, but in general, it covers *y* categories less than scanner B. Due to the naming and classification disparity in vulnerability scanners, the content of the various databases cannot be compared to each other straight away. Everything needs to be standardised first in order to allow a fair comparison. Therefore, a standard set of vulnerability categories is used in the VPA. After the assessment, a report is generated to show the findings. This report can also be seen as the more digested version of the original scan report.

The assessment of the vulnerability products is based only on the content and coverage of its vulnerability database. Performance, stability, user ability and so on, are not considered. The next section shows the conceptual model of the VPA, as well as the various components of the model.

### 7.2.1 A Conceptual Model for Vulnerability Product Assessment

The conceptual VPA model comprises four components and is presented in Figure 7.1.

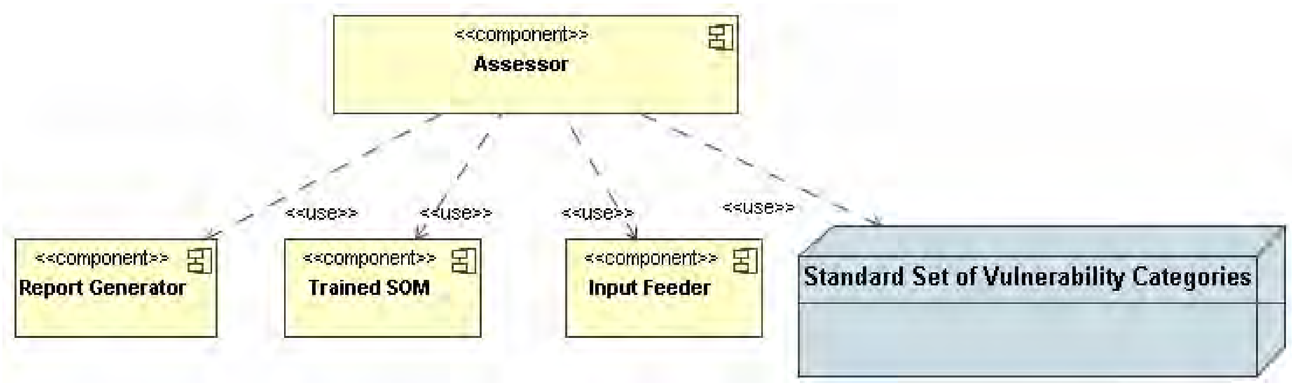


*Figure 7.1 A conceptual model for Vulnerability Product Assessment*

An explanation of the main components in Figure 7.1 follows below.

- Input Feeder

  This component provides users the means to specify input data for the VPA. Once the user has located the input data source, the Input Feeder is responsible for connecting and accessing the data. The input data can either be a vulnerability database (either from a vulnerability scanner or an independent source), or a scan report.

- Standard Set of Vulnerability Categories

  This is the Categorised CVE list generated by the VA as discussed in the previous chapter. It contains all cluster labels for all of the CVE entries in it. This component is used to perform lookup for the category of a specific CVE entry.

- Trained SOM

  This is the fully trained SOM that is generated from the initial categorisation process of the VA. The SOM is used to perform pattern mapping for vulnerabilities that do not contain valid CVE keys.

- Assessor

  This is the main component of the VPA. It collaborates with all other components to perform assessment. As input data is retrieved by the Input Feeder, the Assessor is responsible for classifying the inputs. If the input already contains a CVE reference, then the Assessor will simply do a lookup on the Categorised CVE list to retrieve the category for it. However, if the input does not contain a CVE reference, then the Assessor will transform the input into a numeric vector format, use pattern mapping of the SOM to map the vulnerability to a neuron unit that best matches the numerical input vector and finally retrieve the corresponding category for the mapped neuron unit. The assessment process is explained in more detail in the later sections.

- Report Generator

  Once the assessor has classified all the input data, the overall results are produced and reported to the user. The Report Generator component is responsible for generating such reports. The generated assessment report is shown in Figure 7.5.

The vulnerability database is a database containing signatures of known weaknesses in software or hardware as found in the computing environment. It is either linked to a vulnerability product, or it is managed by an independent institution, for example OSVDB [OSVD 06]. Thus, the more data in the vulnerability database, the more vulnerabilities the tool can detect. The vulnerability database model and structure can differ considerably from one product to another. Refer to Chapter 3 for more details on vulnerability databases.

The next section explains in detail the assessment process of the VPA.

### 7.2.2 Assessment Process

Figure 7.2 shows the activity diagram of the assessment process in the VPA. After the user has specified the input data source, the Input Feeder component will access the data and feed it to the Assessor. While specifying the input data source, the user is going to indicate whether the data source contains CVE keys or not. If it does contain the CVE keys, then the user should specify the column for it. Please note that even if the option "Contains the CVE keys" is selected, it is still possible that a data record may have no CVE key. For each record in the input data source, the following process is performed:

1. If the data record contains a CVE key:
    a. If the CVE key is valid, then look up the CVE key in the Categorised CVE database and get the category assigned for that CVE key.
    b. If the key is invalid, then perform data preprocessing to transform the natural description of the vulnerability data into a numeric vector format and use pattern mapping for the SOM to do the classification.
2. If the data record does not contain a CVE Key:
    a. Do 1b.
3. Update the assessment report.

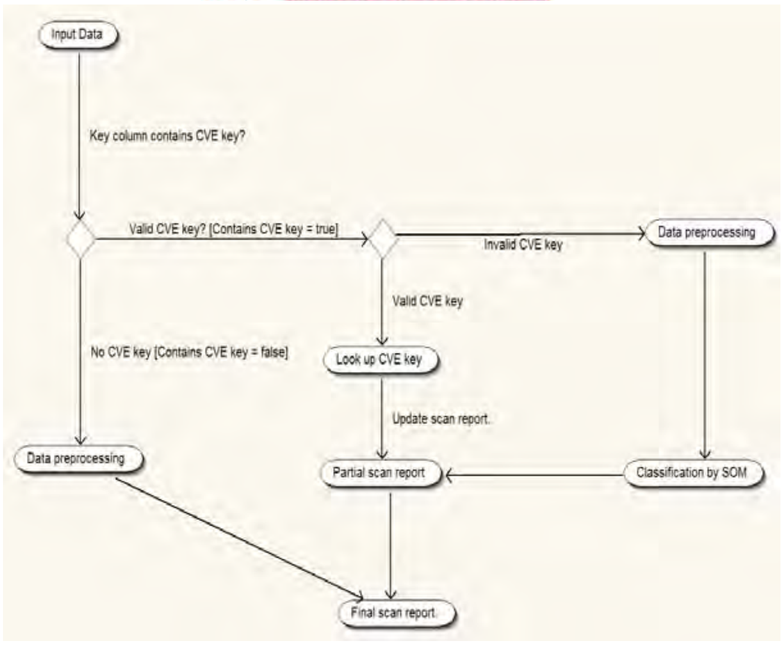


*Figure 7.2 Activity diagram of the VPA*

The following section discusses the VPA prototype that has been developed for this research.

## 7.3 THE VULNERABILITY PRODUCT ASSESSOR (VPA) PROTOTYPE

The VPA prototype implements the VPA specification. Initially, the application only allows the user to connect to a database table (refer to Figure 7.3) by entering the database and table name in the text fields and clicking the **DB Connect** button as shown in Figure 7.3. If the connection is successful, then the check box and combo boxes are enabled, and populated with the table columns. The user must then select the column that describes the vulnerability and the column that contains the key for that vulnerability (refer to Figure 7.4 – the Description, CVE Key and Table Index dropdown lists are populated with the columns loaded from the connecting table). If the vulnerability key contains a CVE reference, then the check box **Contains CVE reference** must be selected. This will allow any CVE references to be mapped directly to the categorised CVE. If the check box is not selected, then the description is used to categorise the vulnerabilities. The final step is to click the **Start** button. This will start the process of categorising the vulnerabilities in the table specified.

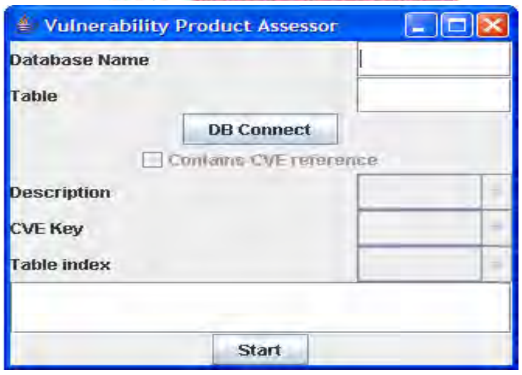
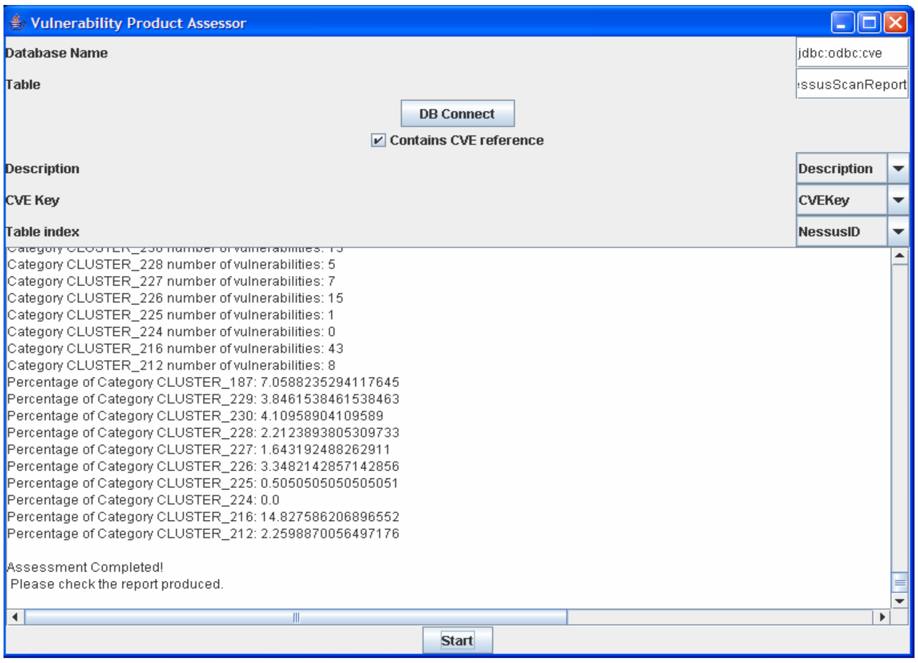


*Figure 7.3 First screen of VPA prototype*



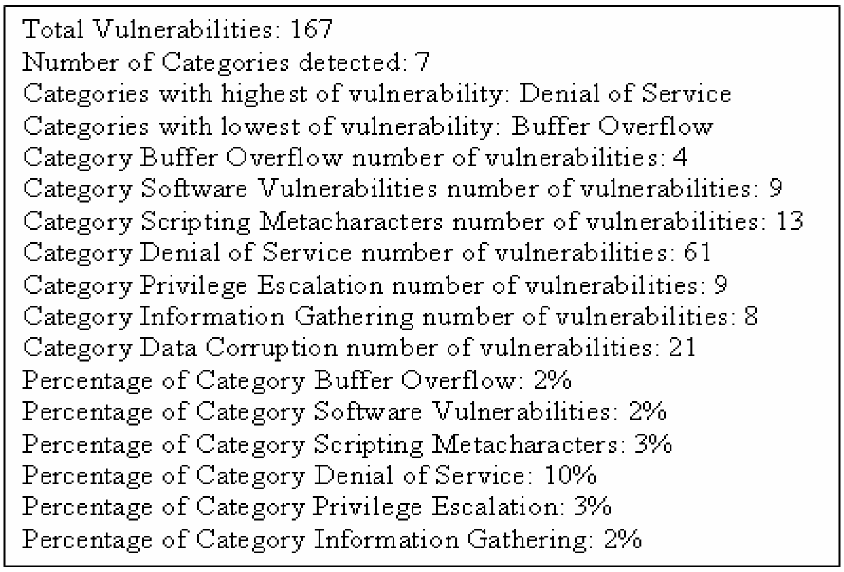
Database Name: jdbc:odbc:cve
Table: ssusScanReport
[✓] Contains CVE reference
Description: Description
CVE Key: CVEKey
Table index: NessusID

```
Category CLUSTER_228 number of vulnerabilities: 5
Category CLUSTER_227 number of vulnerabilities: 7
Category CLUSTER_226 number of vulnerabilities: 15
Category CLUSTER_225 number of vulnerabilities: 1
Category CLUSTER_224 number of vulnerabilities: 0
Category CLUSTER_216 number of vulnerabilities: 43
Category CLUSTER_212 number of vulnerabilities: 8
Percentage of Category CLUSTER_187: 7.0588235294117645
Percentage of Category CLUSTER_229: 3.8461538461538463
Percentage of Category CLUSTER_230: 4.10958904109589
Percentage of Category CLUSTER_228: 2.2123893805309733
Percentage of Category CLUSTER_227: 1.643192488262911
Percentage of Category CLUSTER_226: 3.3482142857142856
Percentage of Category CLUSTER_225: 0.5050505050505051
Percentage of Category CLUSTER_224: 0.0
Percentage of Category CLUSTER_216: 14.827586206896552
Percentage of Category CLUSTER_212: 2.2598870056497176

Assessment Completed!
 Please check the report produced.
```

*Figure 7.4 Database connected*

Figure 7.5 shows an assessment report generated by the VPA for a Nessus scan report. The original Nessus scan report is shown in Appendix C, together with other files generated by the VPA for the scan report. The original Nessus scan report is first transformed into a database table with all relevant attributes, and then assessed by the VPA. As shown in Figure 7.5, the assessment report consists of three sections. The first contains the number of vulnerabilities that were scanned and the number of categories detected in the database. More details can be displayed regarding the vulnerability category with the highest or lowest number of vulnerabilities. The second section displays the details of each vulnerability category that was detected. The last section is a percentage of the number of vulnerabilities in a category that was detected in the scan report, over the number of vulnerabilities that exists for that category in the Categorised CVE database. This report is short and concise, and it highlights the strengths and weaknesses of the product under assessment at a glance.



```
Total Vulnerabilities: 167
Number of Categories detected: 7
Categories with highest of vulnerability: Denial of Service
Categories with lowest of vulnerability: Buffer Overflow
Category Buffer Overflow number of vulnerabilities: 4
Category Software Vulnerabilities number of vulnerabilities: 9
Category Scripting Metacharacters number of vulnerabilities: 13
Category Denial of Service number of vulnerabilities: 61
Category Privilege Escalation number of vulnerabilities: 9
Category Information Gathering number of vulnerabilities: 8
Category Data Corruption number of vulnerabilities: 21
Percentage of Category Buffer Overflow: 2%
Percentage of Category Software Vulnerabilities: 2%
Percentage of Category Scripting Metacharacters: 3%
Percentage of Category Denial of Service: 10%
Percentage of Category Privilege Escalation: 3%
Percentage of Category Information Gathering: 2%
```

***Figure 7.5 Assessment Report for a Nessus Scan Report***

## 7.4 CONCLUSION

Chapter 7 proposed both a conceptual architecture to perform vulnerability product assessment and the comparison of vulnerability products, using the standard set of vulnerability categories generated by the categorisation process. A prototype known as the Vulnerability Product Assessor (VPA), which implements the conceptual architecture, was presented. The implementation and design of the VPA was discussed in detail. The VPA can assess vulnerability products regardless of whether it references the CVE list or not (a vulnerability product can either be a vulnerability scanner or a vulnerability database). In cases where a CVE reference is available, the VPA will simply retrieve its category from a Categorised CVE table. When a CVE reference is missing, the VPA first performs data preprocessing to transform the description into numeric vector format, and then uses the SOM to do pattern mapping that will classify the numeric vector in a category that best matches it. The VPA further demonstrates how to improve other aspects of vulnerability scanners, for example, producing a more summarised report instead of the normal lengthy scan reports. The VPA produces an assessment report at the end of the process. This report highlights the strengths and weaknesses of the vulnerability tool or database under assessment. The report also contains the total number of vulnerability categories that the tool or database can detect, the number of vulnerabilities contained for each category, and a coverage percentage.

The next chapter will conclude with some final remarks about the categorisation process, the VA and VPA prototype implementations, and suggestions for future work.

# CHAPTER 8

# 8 CONCLUSION

## *8.1 INTRODUCTION*

This dissertation represents a methodology for standardising vulnerability categories by means of a data-clustering technique, as well as an application that accesses vulnerability products by using the standardised vulnerability categories. In this chapter, the researcher evaluates the extent to which the objectives of this research study have been achieved. Finally, the researcher suggests further research to be considered.

## *8.2 REVISITING THE PROBLEM STATEMENT*

The principal aim of this research study was to make a contribution towards vulnerability scanning technology. To do so, the researcher set out to develop a methodology to standardise vulnerability categories.

The problems that were to be addressed in the study (stated in Chapter 1 in the form of research questions) are re-examined in the sections that follow with a view to ascertaining the extent to which they have been solved.

### What is the state of current Vulnerability Scanners?

An answer to this research question is attempted in Chapter 2. The architecture and different types of vulnerability scanners available, the current status of this technology and their strengths and weaknesses, are discussed in the chapter. Two types of vulnerability scanning are available, namely network-based scanning and host-based scanning. Network-based scanning discovers and provides rectification information for network-based vulnerabilities. It secures network perimeters and reinforces first lines of defence against attackers. Host-based scanning provides a supplementary level of security by locking down hosts on the network to prevent critical resources from being accessed through internal misuse or external intruders using compromised user accounts. The main strength of this technology is that vulnerability scanners can identify known security flaws before potential intruders do. They discover the vulnerabilities that are on the network, where these vulnerabilities are located and how they can be fixed. The main weaknesses of vulnerability scanners include: false negatives, false positives, a huge administrative burden, and disparities in vulnerability scanners.

## How can the disparity in current vulnerability scanners be solved?

The disparity in vulnerability scanners is one of the major constraints of current vulnerability scanners. As shown in Chapter 3, naming disparity and classification disparity in vulnerabilities have limited collaboration and interoperation among vulnerability products. This research question is the main focus of the present research and it is addressed in Chapter 5, where a methodology is developed to standardise vulnerability categories for known vulnerabilities. The two-phased categorisation process involves a once-off categorisation for existing vulnerabilities and an on-going classification for new vulnerabilities as they get discovered. A prototype implementing this categorisation process – Vulnerability Analyser (VA) – was discussed in Chapter 6 and its results were also critically evaluated in the chapter. The final results of the categorisation presented seven vulnerability categories: Buffer Overflow, Denial of Service, Scripting Metacharacters, Privilege Escalation, Data Corruption, Information Gathering and Software Vulnerabilities.

## How can vulnerability scanners be effectively assessed?

Due to disparity in vulnerability scanners, VS assessment and comparison cannot be carried out in a uniformed way, as there is no real measurement for vulnerability product assessment currently available. Based on standardised vulnerability categories, Chapter 7 of this dissertation presents an application – Vulnerability Product Assessor (VPA) – which facilitates vulnerability product assessment and comparison. It is an extension of the VA prototype that assesses vulnerability databases and scanners. The VPA’s assessment report highlights the strengths and weaknesses of the vulnerability product. Based on this assessment report, vulnerability products can be compared and assessed.

### How can the administrative burden caused by current vulnerability scanners be improved?

Currently, vulnerability scan reports are lengthy and it is difficult and time consuming for administrators to effectively attend to the vulnerabilities reported. It places an immense administrative burden on human resources to rectify them. This problem is again addressed by the VPA. The VPA generates high-level, more digested reports that allow administrators to easily identify shortfalls in their environments and determine macro-level vulnerability patterns. This improves the efficiency and effectiveness of risk management.

## *8.3 FUTURE RESEARCH*

Although the proposed methodology achieved the objectives to the extent described in the section above, it still suffers from certain limitations or has room for improvement. Fortunately these limitations/enhancements provide opportunities to extend and support the work described in this thesis by a number of future research projects as suggested below:

- **Manual Overwrite on categories after categorisation**

  At the moment, the VA prototype contains automated steps only. Vulnerabilities are being clustered by the SOM, and the user cannot change the assigned category in the VA. In future, manual overwrite can be built into the VA so that users can overwrite the VA-assigned category for the vulnerabilities if they feel it to be inappropriate.
- **Automatic updates feed from CVE to Vulnerability Analyser**

  Many new vulnerabilities are being discovered on a daily basis. At present a manual process is required to save new CVE entries into the database and then run the Classification phase of the VA to classify them. In future, an automatic feed can be built between the CVE and VA, so that as soon as something new is published by CVE, it will automatically be fed into VA and trigger the Classification phase.
- **Applying supervised learning or another data-clustering technique**

  A comparison of other data-clustering techniques shall be useful to determine the categories and possibly to cross-validate categories that are generated. Supervised learning may provide benefits by first classifying some of the vulnerabilities and then using a neural network to classify the remaining vulnerabilities.
- **Combine the VPA assessment reports with the original vulnerability database or scan report (provide links to the original text)**

Currently, the VPA assessment reports are independent reports to the original databases or scan reports. As an enhancement, linking the assessment reports to the original scan report or database will provide easy navigation to the users and thus help them to gain a clearer understanding of the assessment reports.

- **Publish standard vulnerability categories online to the public**
  One of the evaluation criteria set out in Chapter 6 is that the resultant categories should be accepted. Publishing the results and the VA online to the public facilitates collaboration with the general public and helps to gain support from industry.
- **Vulnerability Product Parser**
  A manual process is presently required to transform a scan report into a data table format. In future, this can be automated by a generic parser program that will transform any text-based scan reports into its corresponding data table structure.

The CVE list is the de facto standard for vulnerabilities, and vulnerabilities underwent a strict naming process to ensure consistency. The CVE list was selected as the data source for the categorisation process because of this feature. Since the SOM functions on unsupervised learning, it is not influenced by any of the researcher’s opinions and offers a totally unbiased categorisation. The results of this research have shown that using a data-clustering approach to categorise vulnerabilities shows great promise, and the standardising of vulnerability categories will address quite a number of the main shortfalls of Vulnerability Scanner technology.

# REFERENCES

[ACMC 94] ACM, ACM CR Classifications, ACM Computing Surveys 35, pp. 5-16, 1994.

[AGRA 98] Agrawal R., Gehrke J., Gunopulos D. and Raghavan P., Automatic subspace clustering of high dimensional data for data-mining applications. Proceedings of ACM-SIGMOD International Conference on Management of Data, 1998.

[ALIR 04] Ali R., Ghani U. and Saeed A., Data Clustering and its Applications, http://members.tripod.com/asim_saeed/paper.htm, November 25, 2004.

[ANGE 93] Angeline, P.J. and Pollack, J.B., Competitive environments evolve better solutions for complex tasks. In Forrest, S., editor, Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 264-270, San Mateo, CA. Morgan Kaufmann, 1993.

[ANKE 99] Ankerst M., Breunig M., Kriegel H.P. and Sander J., Optics: Ordering points to identify the clustering structure. Proceedings of ACM-SIGMON Conference on Management of Data, 1999.

[BACE 00] Bace R.G., Intrusion Detection, pp. 134-154, Macmillan Technical Publishing, 2000, ISBN 1-57870-185-6.

[BAKE 99] Baker D.W., Christey S.M., Hill W.H. and Mann D.E., The Development of a Common Enumeration of Vulnerabilities and Exposures, Second International Workshop on Recent Advances in Intrusion Detection, 1999.

[BANF 93] Banfield, J.D. and Raftery, A.E., Model-based Gaussian and non-gaussian clustering, 1993.

[BSIB 03] BSI BUSINESS INFORMATION; Information Security; “What is Information Security?”, http://www.bsi-global.com, 2003.

[BUGT 07] Bugtraq Vulnerability database, http://www.securityfocus.com/bid, July, 2007.

[BUGT 06] BugTraq Vulnerability 8390, http://www.securityfocus.com/bid/8390, June 23, 2006.

[CERT 06] Vulnerable products from US-CERT, http://www.us-cert.gov/cas/bulletins/SB06-170.html, June 20, 2006.

[COLE 02] Cole E., Hackers Beware – Defending Your Network from the Wiley Hacker, pp. 238-239, New Riders Publishing, 2002, ISBN 0-7357-1009-0.

[CISC 06] CISCO Systems, http://www.cisco.com/, June 23, 2006.

[CTVU 06] Vulnerability Notes VU#92516, http://www.kb.cert.org/vuls/id/925166, June 20, 2006.

[CVEN 06] CVE Entry 2003-0735, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2003-0735, June 20, 2006.

[DESA 94] De Sa V.R., *Unsupervised Classification Learning from Cross-Modal Environmental Structure*. PhD thesis, Department of Computer Science, University of Rochester, also available as TR 536, 1994.

[DIDA 76] Diday E. and Simon J.C., *Clustering analysis.* Digital Pattern Recognition, Springer-Verlag, pp. 47-94, 1976.

[EAST 96] Easter M., Kriegel H.P., Sander J. and Xu X., *A density-based algorithm for discovering clusters in large spatial databases with noise.* Proceedings of ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining, 1996.

[EDRO 04] MITRE, *CVE Editorial Board Roles, Tasks and Qualifications*, http://www.cve.mitre.org/board/edroles.html, September 29, 2004.

[ENCY 04] Encyclozine, *Internet Protocol Spoofing,* http://encyclozine.com/Internet_protocol_spoofing, June 18, 2004.

[ENGE 02] Engelbrecht A.P., *Computational Intelligence, An Introduction*, pp. 61-71, John Wiley & Sons, Inc., ISBN 0-470-84870-7, 2002.

[ENTS 02] ENT, *Intrusion Detection and Vulnerability Testing tools: What works? – Technology Information*, http://articles.findarticles.com/p/articles/mi_m0FOX/is_19_5/ai_68644208, November 2002.

[FBIR 06] Federal Bureau of Investigation Survey, http://www.abovesecurity.com/doc/CommuniquesPDF/FBISurvey2006.pdf, December 2006.

[FISH 89] Gennari, J.H., Langley, P. and Fisher, D. Models of incremental concept formation. *Artificial Intelligence* 40-1(3), pp. 11-61, 1989.

[FREE 06] FreeBDS, http://www.freebsd.org/, June 12, 2006.

[GOLL 99] Gollmann G., *Computer Security*, pp. 5-9, John Wiley & Sons, Inc., ISBN0-471-97844-2, 1999.

[HANJ 00] Han J. and Kamber M., *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[HANK 01] Han J. and Kamber M., *Data Mining: Concepts and Techniques*, pp. 105-130, Morgan Kauffmann Publishers, ISBN 1-55860-489-8, 2001.

[HARN 02] Harnist E., *Intelligent Vulnerability Scanner*, http://www.giac.org/practical/Eric_Harnist_GSEC.doc, January 2002.

[IFAC 98] INTERNATIONAL FEDERATION OF ACCOUNTANTS, *Managing Security of Information*, International Information Technology Guideline, ISBN 1-887-464-31-X, 1998.

[ISSC 06] Internet Scanner, http://www.iss.net/products_services/enterprise_protection/vulnerability_assessment/scanner_internet.php, August 25, 2006.

[ISSS 04] Internet Security Systems, Network and Host-based Vulnerability Assessment, http://www.isskk.co.jp/customer_care/resource_center/whitepapers/nva.pdf, June 18, 2004.

[ISTR 07] Symantec, *Internet Security Thread Report*, http://www.symantec.com/enterprise/theme.jsp?themeid=threatreport, July 2007

[JAIN 93] Jain A. and Flynn P.J., *Three-Dimensional Object Recognition Systems.* Elsevier Science Inc., New York, NY, 1993.

[JAIN 99] Jain A.K., Murty M.N. and Flynn P.J., *Data clustering: A Review*, ACM Computing Surveys, Vol. 31, No. 3, September 1999.

[KAUF 90] Kaufman L. and Rousseeuw P.J., *Finding croups in data: an introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[KIRK 01] Kirk J.S., Chang D.J. and Zurada J.M., *A Self-Organizing Map with Dynamic Architecture for Efficient Color Quantization*, Proceedings of the International Joint Conference on Neural Networks, Washington, D.C., 2001.

[KOHO 95] Kohonen T., *Self-Organizing Maps*, Berlin, Germany: SpringerVerlag, 1995.

[KUJA 03] Kujawski P., *Why Networks Must Be Secured*, Cisco Systems, Inc., 2003.

[LIYU 04] Li Y.L., Venter H.S. and Eloff J.H.P, *Categorizing vulnerabilities using data-clustering techniques*, Proceedings of ISSA Conference, June 2004.

[LOPY 01] *Enterprise-Wide Network Vulnerability Assessment,* http://www.sans.org/rr/audit/scan_model.php, 2001.

[LING 06] Ling J. and Elert G., *Number of words in the English Language*, http://hypertextbook.com/facts/2001/JohnnyLing.shtml, June15, 2006.

[MACQ 67] MacQueen J., *Some methods for classification and analysis of multivariate observations,* Proceedings of 5th Berkeley Symp. Math. Statist. Prob., 1967.

[MAIW 03] Maiwald E., *Network Security: A Beginner's Guide*, Second Edition, "Defining Information Security", p. 4, Osborne McGraw-Hill, ISBN 0-07-2229-578, 2003.

[MANN 99] Mann D. and Christey S.M., *Towards a Common Enumeration of Vulnerabilities*, 2nd Workshop on Research with Security Vulnerability Databases, 1999.

[MART 03] Martin R., Integrating Your Information Security Vulnerability Management Capabilities Through Industry Standards (CVE & OVAL), IEEE, 2003.

[MCCL 02] McClure S., Scambray J. and Kurtz G., *Hacking Exposed,* pp. 581-582, McGraw-Hill/Osborne, ISBN 0-07-219382-4, 2002.

[MICH 83] Michalski R., Stepp R.E. and Diday E., *Automated construction of classifications: conceptual clustering versus numerical taxonomy,* IEEE Trans. Patten Anal. Mach. Intell., PAMI-5, 5, pp. 396-409, September 1983.

[MICR 02] Microsoft Commerce Server 2002 – The STRIDE THREAT MODEL, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csvr2002/htm/cs_se_securecode_zlsj.asp May 15, 2004.

[MITR 06] MITRE Inc., http://www.cve.mitre.org/, April 27, 2006.

[MORE 04] MOREnet, http://www.more.net/services/rva/categories.html, May 2004.

[MOXO 96] Moxon B., *Defining Data Mining, The Hows and Whys of Data Mining, and How it differs from other analytical techniques*, Online Addition of DBMS Data Warehouse Supplement, August 1996.

[MURT 95] Murty M.N. and Jain A.K., *Knowledge-based clustering scheme for collection management and retrieval of library books*, Pattern Recognition 28, pp. 949-964, 1995.

[MURZ 95] Murzin A., Brenner S., Hubbard T. and Chothia C., SCOP: a structural classification of proteins database for the investigation of sequences and structures, *Journal of Molecular Biology*, 247, pp. 536-540, 1995.

[MUTC 02] Mutch D.M., Berger A., Mansourian R., Rytz A. and Roberts M.A., *The limit fold change model: a practical approach for selecting differentially expressed genes from microarray data,.* BMC Bioinfomatics, 2002.

[NESS 06] Nessus, http://www.nessus.org/ , March 28, 2006.

[NESP 06] Nessus plugin id 12213, http://www.nessus.org/plugins/index.php?view=single&id=12213, April 03, 2006.

[NETR 06] NetRecon, http://www.symantec.com/region/can/eng/product/netrecon/, March 28, 2006.

[NETW 04] Network World Fusion, *Vulnerability Assessment tool,* http://www.nwfusion.com/, May 05, 2004.

[NESP 06] Nessus plugin id 12213, http://www.nessus.org/plugins/index.php?view=single&id=12213, 3 April 2006.

[OSMA 05] Osma R.Z., Foss A., Lee C. and Wang W., *On data clustering analysis: Scalability, constraints and validation*, 2005.

[OSID 06] OSVDB entry 2410, http://www.osvdb.org/displayvuln.php?osvdb_id=2410, June 20, 2006.

[OSVD 06] Open Source Vulnerability Database, http://www.osvdb.org/, May 05, 2006.

[OSCT 06] OSVDB categories, http://www.osvdb.org/search.php, June 20, 2006.

[PEDE 97] Pedersen T. and Bruce R., *Distinguishing word senses in untagged text*, Proceedings of the Second Conference on Empirical Methods in Natural Language Processing, pp. 197-207, 1997.

[PFLE 03] Pfleeger C.P., *Security in Computing*, Prentice Hall, ISBN 0-13-035548-8, 2003.

[QUAL 06] QualysGuard, http://www.qualys.com/products/qgen/, March 28, 2006.

[RETI 06] Retina Network Security Scanner, http://www.eeye.com/html/Products/Retina/index.html, March 2006.

[RICH 01] Richard O. Duda, Peter E. Hart, David G. Stork, *Pattern classification* (2nd edition), Wiley, New York, ISBN 0471056693, 2001.

[SAIN 06] SAINT Corporation, http://www.saintcorporation.com, March 2006.

[SCHA 79] Schacher B.J., Davis L.S. and Rosenfeld A., *Some experiments in image segmentation by clustering of local feature values,* Pattern Recognition 11, pp. 19-28, 1979.

[SCHN 99] Schneier B., *Security is Not a Product; It's a Process*, Crypto-Gram Newsletter, http://www.counterpane.com/crypto-gram-9912.html#SecurityIsNotaProdcutItsaProcess, December 1999.

[SCHN 00] Schneier B., *Secrets and Lies, Digital Security in a Networked World*, pp. 194-197, John Wiley & Sons, Inc., ISBN 0-471-25311-1, 2000.

[SECU 05] SecurityFocus, http://www.securityfocus.com, May 31, 2005.

[SFPR 04] SFProtect, http://www.winnetmag.com/Article/ArticleID/8401/8401.html, April 28, 2004.

[SHEI 98] Sheikholeslami G., Chatterjee S. and Zhang A., *Wavecluster: a multi-resolution clustering approach for very large spatial database,* Proceedings of 24th Conference on Very Large Databases, 1998.

[SILV 88] Silverman J.F. and Cooper D.B., *Bayesian clustering for unsupervised estimation of surface and texture models,* IEEE Trans. Pattern Anal. Mach. Intell. 10, pp. 482-495, July 1988.

[SNYD 03] Snyder J., *How Vulnerable*, http://infosecuritymag.techtarget.com/2003/mar/cover.shtml, March 2003.

[TIWA 00] Tiwana A., *Web Security,* pp. 112-135, Digital Press, ISBN 1-55558-210-9, 2000.

[VENT 02] Venter, H.S. and Eloff, J.H.P., Harmonising Vulnerability Categories*, South African Computer Journal;* pp. 24-31; No. 29; Computer Society of South Africa, ISSN 1015-7999, 2002.

[VENT 03] Venter H.S., A Model For Vulnerability Forecasting, PhD thesis, Faculty of Natural Sciences, Rand Afrikaans University, 2003.

[WEBO 04] WEBOPEDIA, http://www.webopedia.com/TERM/V/vulnerability_scanning.html, April 24, 2004.

[ZENE 04] Zenero S. and Savaresi S.M., *Unsupervised learning techniques for an intrusion detection system*, Proceedings of SAC 04 Conference, Nicosia, Cyprus, March 2004.

# APPENDIX A – Insignificant Words.txt

An extract of the file is shown here, and the complete Insignificant Words.txt file can be found on the CD attached /Dissertation/Appendix.

```
'
$
$$
$_phplib[libdir]
$attach$
$b
$hostinput
$password
%
%%
%c
%d
%m
%s
%systemdrive%
&
&={script}
*
.
/
:
::$data
?
?&
??
?wp-cs-dump
?wp-ver-info
[
[b]
\
]
^d
_
{
|
~
~!
~{
+
++
<
<@>
<<all
<If>
<script>
<soundname>
=
>
>=
-
-i
-g
-bit
-beta
-path
-display
-file
-po
-pl
-fuser
-xkbmap
-dev
-zi
-mode
-st
-dsafer
-config
-tn
-froot
-restore_config
-byte
-dallow_updates
-rh
-pre
-msql
-text
-t
-m
-e
-s
-l
-p
-h
-d
-f
-j
-n
-o
-q
-u
-w
-x
-enabled
-encoded
--config
--fuser
_conv_principal
_init
_private
-research
-rcfile
_tt_transaction
_mailto
<cr>
'
**@target
_proxy
_tt_create_file
_writers
_rdp
_basket
!nick
~root
_conv_principal
_mime_split
_tt_create_file
<cr>
<lf>
a
a-
aaa
able
abnormal
abor
abort
aborts
about:
above
absent
absolute
abuse
abyssws
acc
accelerated-x
accept
accept_filter
accept_fw
accepted
accepting
accepts
accessed
accesses
accessible
accessing
access-request
accounting
acd
ace
acme
across
acros-suencksen
act
actionpoll
activate
active
activeperl
activestate
activites
activity
actual
actually
ad
adcycle
add
add_
addauthor
addbuddy
added
addgame
adding
addition
additional
addressed
adds
addsulog
addview
adi
adk
adlogin
adm
admin$
admin_password
adminpassword
adminusers
adobe
adsl
adtran
advanced
advancestack
advertisement
advertisements
advfs
affects
afpacache
afstokenpassing
after
again
against
aglimpse
aid
aim
aio
aironet
aka
```

# APPENDIX B – Significant Word Set

| UniqueWords |
|---|
| access |
| account |
| accounts |
| acl |
| activex |
| activities |
| address |
| addresses |
| administration |
| administrative |
| administrator |
| agent |
| aix |
| allaire |
| antivirus |
| apache |
| apple |
| applet |
| application |
| applications |
| argument |
| arguments |
| arp |
| asp |
| attachment |
| attacker |
| attribute |
| authentication |
| authorization |
| beta |
| bind |
| broadcast |
| browser |
| brute |
| bsd |
| bsd-based |
| buffer |
| bug |
| bugs |
| bugzilla |
| bypass |
| cache |
| caldera |
| call |

| UniqueWords |
|---|
| capability |
| card |
| cart |
| cd |
| certificate |
| cgi |
| cisco |
| class |
| cleartext |
| code |
| coldfusion |
| com |
| communicator |
| compaq |
| component |
| components |
| computer |
| concentrator |
| condition |
| conf |
| configuration |
| configurations |
| connection |
| connections |
| console |
| consumption |
| content |
| controls |
| cookie |
| cookies |
| core |
| cpu |
| credentials |
| crontab |
| cross-site |
| cups |
| cwd |
| daemon |
| data |
| database |
| debian |
| denial |
| descriptor |
| desktop |

| UniqueWords |
|---|
| dev |
| device |
| devices |
| direct |
| directories |
| directory |
| display |
| dll |
| dns |
| document |
| domain |
| domino |
| dump |
| echo |
| edition |
| email |
| e-mail |
| emulator |
| encryption |
| engine |
| enterprise |
| entries |
| entry |
| excel |
| exe |
| execution |
| exhaustion |
| explorer |
| express |
| extension |
| extensions |
| feature |
| field |
| file |
| filename |
| firewall |
| firewall- |
| firmware |
| folder |
| folders |
| form |
| format |
| frame |
| freebsd |

# APPENDIX C – Sample Nessus Scan Report

A sample Nessus Scan report can be found on the CD attached /Dissertation/Appendix.