This is to certify that the
thesis entitled

CUSTOMIZABLE VULNERABILITY ANALYSIS AND
CLASSIFICATION

presented by

BRENT LEE HOLTSCLAW

has been accepted towards fulfillment
of the requirements for the

M.S. degree in COMPUTER SCIENCE

Major Professor's Signature

May 8, 2008

Date

CUSTOMIZABLE VULNERABILITY ANALYSIS AND CLASSIFICATION

By

Brent Lee Holtsclaw

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for a degree of

MASTER OF SCIENCE

Department of Computer Science

2008

ABSTRACT

CUSTOMIZABLE VULNERABILITY ANALYSIS AND CLASSIFICATION

By

Brent Lee Holtsclaw

Due to the many complications within the various vulnerability databases, my thesis presents a tool, VACT, which scans the vulnerability databases, searches for specified vulnerabilities by the classification given, and returns the selected vulnerabilities in a downloadable and statistical form. VACT allows for one to gather customizable trend analysis results from a user defined search of vulnerability databases. The user selects each classification used within the search. The search is conducted by comparing the classification to the vulnerabilities description. Trend analysis results are returned by separating the statistics of the vulnerabilities found per year selected.

ACKNOWLEDGEMENTS

I would like to thank my parents for all of the support that I have received while obtaining my advanced degree. I am lucky to have such wonderful parents. In addition, I would like to thank my family for all of the encouragement that I received. A special thank you goes out to Caity.

I would like to thank Dr. Enbody. It has been a real pleasure having him as an advisor and professor. Early in my academic career, his introduction to programming class helped to draw me into computer science. His patient, straightforward approach to teaching and advising has been very beneficial. I could not have completed my thesis without all of his knowledge and help. I would also like to thank the faculty within the Computer Science Department for all of the knowledge that I gained as a result of their teaching.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

My thesis generates trend analyses from user-defined searches of online vulnerability databases. There are many challenges in extracting trends from the various vulnerability databases. Vulnerability databases vary on which vulnerability information is kept within the database. They also differ on the way that the data is classified and how one is allowed to search through the data. In addition, no vulnerability database offers trend analysis on a user-defined search. To help provide consistency within these factors, the thesis presents a tool, Vulnerability Analysis and Classification Tool (VACT), which combines vulnerability database search with trend analysis tools to enhance the ability of the end user to search through vulnerabilities and conduct analysis.

There are many different vulnerability databases that exist to help raise awareness of the various know vulnerabilities. The government runs some while private organizations or universities run others. Each database is set up with different standards and capabilities. Take US-CERT for example, the Vulnerability Notes Database contains only severe vulnerabilities. ("US-CERT Vulnerability Notes Database.") On the other hand, Secunia contains advisories and virus information. ("Search Advisory, Vulnerability, and Virus Database.") Just as each database contains its own set of vulnerabilities, there are multiple vulnerability schemes to fulfill the various needs of researchers, developers, and systems administrators which provides an interesting scenario when searching for vulnerabilities to suit an individual's needs. Therefore, one goal is to provide users with a tool that allows a customizable search to harvest the desired vulnerabilities.

Browsing and searching are two main ways that one is able to use in order to find specific vulnerabilities within a vulnerability database. Neither feature follows a universal standard, but they use similar concepts. When browsing a database one is able to view vulnerabilities according to some kind of vulnerability classification that is defined within the database. The classification schema allows the vulnerability databases to presort vulnerabilities by common characteristics. The search feature acts like a search engine allowing the users to input a search string to find within the various vulnerabilities. The variance within searches comes from the way that the vulnerability database searches for vulnerabilities. For example, Open Source Vulnerability Database will search for the string within titles while USCERT will search for the keyword within the vulnerability's information. ("OSVDB: The Open Source Vulnerability Database.") To take away ambiguity when searching for certain characteristics, the Vulnerability Analysis and Classification Tool allows a user the ability to not only search by key words but to also make up a classification. Through our own trials, we have found that an efficient way to return these results is to do a string search using vulnerability descriptions.

To improve the effectiveness of the tool, trend analysis from the searched results is provided. This is a feature missing from many vulnerability databases. The one exception comes within National Vulnerability Database which is sponsored by the National Institute of Science and Technology. ("National Vulnerability Database.") Although it will give some trend analysis pertaining to the vulnerabilities, it does not allow the user any freedom. All analysis must be picked from predefined classifications, which can be improperly calculated at times. Some organizations which are not vulnerability databases, such as Common Weakness Enumeration, also deliver a trend analysis of vulnerabilities. The analysis done by CWE is done by the year and cannot be customized. Our main goal of trend analysis is to return graphs and statistics that can help the user better visualize the results and save the user from any low level computation.

The thesis begins by telling a quick story that motivated this project. We then provide background information about vulnerability classification. Next we provide the reader with insight into the overall capabilities of the tool by providing a breakdown of the steps and decisions that were taken when making the tool. Once we show what the tool does, we then explain how it differs from the current offerings of vulnerability searches and trend analyses. The focus is then shifted to detail the methodologies used to determine both the search and trend analysis results returned. We then describe a real life scenario and how our tool could help out. To round things out the results of some scenarios posed within the search and trend analysis section are put forward as well as the results to the real life scenario. Finally the thesis is finished up with a conclusion which pulls together all of ideas within the thesis.

Chapter 1: Purpose/Background

1.1 Motivation

The thesis was motivated by a simple question posed by Dr. Enbody to classify the number of buffer-type vulnerabilities that occurred over the past year. He specifically wanted to obtain the number of buffer vulnerabilities, the number of compromising buffer vulnerabilities, and the number of compromising vulnerabilities. Compromising vulnerabilities were defined as vulnerabilities that would allow a user to gain control of a system or gain elevated privileges within a system. The first issue arose when we tried to find what classification techniques are used to classify vulnerabilities. After searching the literature and examining vulnerability databases online, we found that there are many different classification schemes. Not only did we not see a common classification for vulnerabilities, but we also found discrepancies within vulnerability databases. As we sorted through the initial problems, we found another problem with the data. We needed to sort through the results of the vulnerabilities to compute our own statistics and graphs. As this simple task continued to grow in complexity, it was apparent that a tool to help automate this process would be a worthwhile contribution to the community.

1.2 Vulnerability Classification Techniques

The word vulnerability has a very strong tone with it as it dictates that a flaw is evident within the subject at hand.

- Within computer security, the term does not change meaning as it applies to a "set of transitions which take a system from an allowed state to a disallowed state" (Bishop)

Another key term that will come into play later in the thesis is an exploit.

- An exploit is a set of commands which take advantage of a vulnerability. (Engle)

Due to the complexity and abundance of various vulnerabilities and exploits that exist, vulnerability databases have been created by various entities to help share the knowledge with users.

Vulnerability Classification

Name	Abbreviation
Bishop	
Common Weakness Enumeration	CWE

Table 1.1. Vulnerability classification ideas

Tables 1.1 and 1.2 are provided to help distinguish the concepts and databases presented throughout chapter one. Table 1.1 contains the current vulnerability classification techniques and ideas. Table 1.2 contains the vulnerability databases introduced within the chapter. It also contains the vulnerability identifications that are used. The various databases contain vulnerabilities according to various naming conventions and standards which are recognized by the vulnerability identifications.

Vulnerability Identification

Name	Abbreviation
Common Vulnerabilities and Exposers	CVE
Bugtraq	
Internet Security Systems' X-Force organization	ISS X-Force
Nessus Script	
Open Source Vulnerability Database	OSVDB
Snort Signature	
Secunia Advisory	
French Security Incidence Response Team	FrSIRT Advisory
Open Vulnerability and Assessment Language	OVAL
Computer Incident Advisory Capability	CIAC Advisory
Computer Emergency Response Team	CERT
The United States Computer Emergency Readiness Team	CERT VU
Milw0rm	
Common Configuration Enumeration	CCE
Common Vulnerability Scoring System	CVSS

Vulnerability Databases

Name	**Abbreviation**	**Searchable Identification**
Bugtraq		
Microsoft Bulletins		
French Security Incidence Response Team	FrSITR	
US-CERT Vulnerability Note Database	US-CERT	CVE, CERT VU
National Vulnerabilities Database	NVD	CVE, CCE, CVSS
Open Source Vulnerability Database	OSVDB	CVE, OSVDB, Bugtraq, ISSX-Force, Nessus Script, Snort Signature, FrSIRT Advisory, OVAL, CIAC Advisory, CERT, CERT VU, Milw0rm
Secunia		CVE, Secunia

Table 1.2. List of Abbreviations associated with vulnerability databases and identifications.

Many key terms and ideas within this thesis hinge on the previous work that was accomplished by Matt Bishop. Bishop a professor at University of California, Davis, specializes in computer security and vulnerability analysis. In a paper from 1999, Bishop defines five important properties to vulnerability classification:

> 1. Similar vulnerabilities are classified similarly. For example, all vulnerabilities arising from race conditions should be grouped together. However, we do *not* require that they be distinct from other vulnerabilities. For example, a vulnerability involving a race condition may require untrusted users having specific access permissions on files or directories. Hence it should also be grouped with a condition for improper or dangerous file access permissions.
>
> 2. Classification should be primitive. Determining whether a vulnerability falls into a class requires a "yes" or "no" answer. This means each class has exactly *one* property. For example, the question "does the vulnerability arise from a coding fault or an environmental fault" is ambiguous; the answer could be either, or both. For our scheme, this question would be two distinct questions: "does a coding fault contribute to the vulnerability" and "does an environmental fault contribute to the vulnerability." Both can be answered "yes" or "no" and there is no ambiguity to the answers.
>
> 3. Classification terms should be well-defined. For example, does a "coding fault" arise from an improperly configured environment? One can argue that the program should have checked the environment, and therefore an "environmental fault" is simply an alternate manifestation of a "coding fault." So, the term "coding fault" is not a valid classification term.
>
> 4. Classification should be based on the code, environment, or other technical details. This means that the social cause of the vulnerability (malicious or simply erroneous, for example) are not valid classes. This requirement eliminates the speculation about motives for the hole. While valid for some classification systems, this information can be very difficult to establish and will not help us discover new vulnerabilities.
>
> 5. Vulnerabilities may fall into multiple classes. Because a vulnerability can rarely be characterized in exactly one way, a realistic classification scheme must take the multiple characteristics causing vulnerabilities into account. This allows some structural redundency in that different vulnerabilities may lie in the same class; but as indicated in 1, above, we expect (and indeed desire) this overlap.(Bishop)

These properties help to provide a straightforward way of creating vulnerability classifications. The same properties are reiterated in another paper in which Bishop was an author seven years later.(Engle)

Although Bishop presents a great plan for vulnerability classification, when one really evaluates the classification rubric, it presents itself as a guideline rather than as specific classifications. Various interpretations can make classifications that are both broad and specific. One will find an assortment of interpretations when searching through the various classifications that are used within the different vulnerability databases.

In addition to the work presented by Bishop, two branches of The MITRE Corporation help to provide structure to vulnerability classification. The Common Weakness Enumeration, CWE, division of The MITRE Corporation, offers a community-developed dictionary of software weakness types. ("Common Weakness Enumeration.") While Common Vulnerability Enumeration, CVE provides a common naming convention for vulnerabilities. ("Common Vulnerabilities and Exposures.")

With the help of CVE and researchers, CWE continues to build a classification tree of vulnerabilities. Even though there is a current layout and structure, CWE continually accepts new research and vulnerabilities to help expand the tree to make it as comprehensive as possible. The layout is "currently using what could roughly be described as a three-tiered approach, in which (1) the lowest level consists of the full CWE List (hundreds of nodes) that is primarily applicable to tool vendors and detailed research efforts; (2) a middle tier consists of descriptive affinity groupings of individual CWEs (25-50 nodes) useful to software security and software development practitioners; and (3) a more easily understood top level consisting of high-level groupings of the middle-tier nodes (5-10 nodes) to define strategic classes of vulnerabilities and which is useful for high-level discourse among software practitioners, business people, tool vendors, researchers, etc."("Process.")

CWE produces a body of work that is the closest we have seen completely implementing Bishops properties. The problem is that it is not used as a database standard. Several databases such as National Vulnerability Database have implemented a partial CWE list. Figures 1.1 and 1.2 illustrate the level of detail that exists within CWE. Figure 1.1 is a high level classification with an associated description and relationships to similar classifications. Figure 1.2 illustrates a specific classification. The classification levels can also be seen within Figures 1.1 and 1.2. Figure 1.2 is referred to as a child of Figure 1.1. In addition to the description and relationships, Figure 1.2 has associated examples. Figures 1.3 and 1.4 illustrate CWE tier 2 and tier 3. The tiers represent the various levels of classification that exists within CWE.

Failure to Constrain Operations within the Bounds of an Allocated Memory Buffer

Weakness ID 119 *(Weakness Class)* **Status:** Draft

Description

Summary

The software may potentially allow operations, such as reading or writing, to be performed at addresses not intended by the developer.

Extended Description

When software permits read or write operations on memory located outside of an allocated range, an attacker may be able to access/modify sensitive information, cause the system to crash, alter the intended control flow, or execute arbitrary code.

Affected Resource Memory

Relationships

Nature	Type	ID	Name
ChildOf	W	118	Range Errors
ChildOf	V	635	Weaknesses Used by NVD
ChildOf	C	633	Weaknesses that Affect Memory
ParentOf	C	133	String Errors
ParentOf	W	123	Write-what-where Condition
ParentOf	W	124	Boundary Beginning Violation ('Buffer Underwrite')
ParentOf	W	125	Out-of-bounds Read
ParentOf	W	128	Wrap-around Error
ParentOf	W	129	Unchecked Array Indexing
ParentOf	W	131	Incorrect Calculation of Buffer Size
ParentOf	W	132	Miscalculated Null Termination
ParentOf	W	466	Return of Pointer Value Outside of Expected Range
ParentOf		120	Unbounded Transfer ('Classic Buffer Overflow')

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
100	Overflow Buffers
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow
42	MIME Conversion
24	Filter Failure through Buffer Overflow
8	Buffer Overflow in an API Call
44	Overflow Binary Resource File
9	Buffer Overflow in Local Command-Line Utilities
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion

Figure 1.1. A snapshot of a node within CWE tier 1

Incorrect Calculation of Buffer Size

Weakness ID 131 *(Weakness Class)* **Status:** Draft

Description

Summary

The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.

Observed Examples

Reference	Description
CVE-2004-1363	substitution overflow: buffer overflow using environment variables that are expanded after the length check is performed
CVE-2004-0747	substitution overflow: buffer overflow using expansion of environment variables
CVE-2005-2103	substitution overflow: buffer overflow using a large number of substitution strings
CVE-2005-3120	transformation overflow: product adds extra escape characters to incoming data, but does not account for them in the buffer length
CVE-2003-0899	transformation overflow: buffer overflow when expanding ">" to ">", etc.
CVE-2001-0334	expansion overflow: buffer overflow using wildcards
CVE-2001-0248	expansion overflow: long pathname + glob = overflow
CVE-2001-0249	expansion overflow: long pathname + glob = overflow
CVE-2002-0184	special characters in argument are not properly expanded
CVE-2004-0434	small length value leads to heap overflow
CVE-2002-1347	multiple variants
CVE-2005-0490	needs closer investigation, but probably expansion-based
CVE-2004-0940	needs closer investigation, but probably expansion-based

Context Notes

This is a broad category. Some examples include: (1) simple math errors, (2) incorrectly updating parallel counters, (3) not accounting for size differences when "transforming" one input to another format (e.g. URL canonicalization or other transformation that can generate a result that's larger than the original input, i.e. "expansion").

This level of detail is rarely available in public reports, so it is difficult to find good examples.

Relationships

Nature	Type	ID	Name
ChildOf	W	119	Failure to Constrain Operations within the Bounds of an Allocated Memory Buffer

Source Taxonomies

PLOVER - Other length calculation error

Applicable Platforms

C

C++

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
47	Buffer Overflow via Parameter Expansion

Figure 1.2. Another screenshot of CWE tier 1

- Location - *(1)*
  - Configuration - *(16)*
  - Code - *(17)*
    - Source Code - *(18)*
      - Data Handling - *(19)*
      - Security Features - *(254)*
      - Time and State - *(361)*
      - Error Handling - *(388)*
      - Failure to Fulfill API Contract (aka 'API Abuse') - *(227)*
      - Use of Inherently Dangerous Function - *(242)*
      - Indicator of Poor Code Quality - *(398)*
      - Insufficient Encapsulation - *(485)*
      - Always-Incorrect Control Flow Implementation - *(670)*
      - Insufficient Control Flow Management - *(691)*
    - Byte/Object Code - *(503)*
      - Compiler Removal of Code to Clear Buffers - *(14)*
    - Violation of Secure Design Principles - *(657)*
      - Design Principle Violation: Failure to Use Least Privilege - *(250)*
      - Design Principle Violation: Not Failing Securely - *(636)*
      - Design Principle Violation: Not Using Economy of Mechanism - *(637)*
      - Design Principle Violation: Not Using Complete Mediation - *(638)*
      - Design Principle Violation: Insufficient Compartmentalization - *(653)*
      - Design Principle Violation: Reliance on a Single Factor in a Security Decision - *(654)*
      - Design Principle Violation: Failure to Satisfy Psychological Acceptability - *(655)*
      - Design Principle Violation: Reliance on Security through Obscurity - *(656)*
      - Design Principle Violation: Lack of Administrator Control over Security - *(671)*
  - Environment - *(2)*
    - Technology-specific Environment Issues - *(3)*
      - J2EE Environment Issues - *(4)*
      - .NET Environment Issues - *(519)*
- Motivation/Intent - *(504)*

Figure 1.3. A snapshot of the upper portion of CWE tier 2

- Location - *(1)*
  - Configuration - *(16)*
  - Code - *(17)*
  - Environment - *(2)*
- Motivation/Intent - *(504)*
  - Intentionally Introduced Weakness - *(505)*
  - Inadvertently Introduced Weakness - *(518)*

Figure 1.4. A snapshot of CWE tier 3

Common Vulnerabilities and Exposures, CVE, "is a list of information security vulnerabilities and exposures that aims to provide common names for publicly known problems. The goal of CVE is to make it easier to share data across separate vulnerability capabilities (tools, repositories, and services) with this 'common enumeration.'" ("Common Vulnerabilities and Exposures.") In other words CVE provides a common naming convention to reference vulnerabilities.

CVE does not include any zero day vulnerabilities. A zero day vulnerability is a newly released vulnerability. Instead, vulnerabilities must go through a process to get onto the CVE list. After a vulnerability is discovered, it goes through "three stages: the initial submission stage, the candidate stage, and the entry stage." ("How We Build the CVE List.") Their website provides a complete tutorial on the CVE List building process.

The CVE List building process is stringent with the vulnerabilities which are given a common name. To encompass vulnerabilities without CVE names, other organizations offer identification to vulnerabilities which are identified within CVE and ones that are not. Security Focus features a zero day vulnerability database, bugtraq. The database allows users to send in all vulnerabilities that are found. Bugtraq offers an up-to-date email system to provide all subscribers a chance to view and discuss new vulnerabilities. Microsoft Bulletins features Microsoft specific vulnerabilities. French Security Incidence Response Team, FrSITR, also keeps a zeroday list of reported vulnerabilities. USCERT provides a truncated list of vulnerabilities by selecting only the vulnerabilities which are identified as critical. Secunia offers a vulnerability list which includes both vulnerabilities and virus information. It is important to note that in addition to their own naming convention, each of the databases listed above still offers the CVE name for vulnerability which is identified by compiles to CVE standards.

Other notable standards which are used by some of the vulnerability databases include the Common Vulnerability Scoring System, CVSS, and Common Configuration Enumeration, CCE. "The Common Vulnerability Scoring System provides an open framework for communicating the characteristics and impacts of IT vulnerabilities. Its quantitative model ensures repeatable accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the scores." ("NVD Common Vulnerability Scoring System Support v2.") CVSS is calculated with using the following metrics: Vulnerability Severity, Access Vector, Authentication, Confidentiality, Integrity, Availability, and Access Complexity. "The CCE List provides unique identifiers to security-related system configuration issues in order to facilitate fast and accurate correlation of configuration data across multiple information sources and tools." ("About CCE.")

1.3 Vulnerability Databases

Within the next section we are going to take a closer look into some of the vulnerability databases. Not only does this help one get a better picture of the types of vulnerability databases that exist but it also shows how vulnerability databases differ. The description will feature what types of vulnerabilities that can be found within the database as well as any classification schemes that are used. Another attribute of the description will be an evaluation of the search functionality within the database.

The US-CERT Vulnerability Notes Database contains vulnerabilities that meet a "certain severity threshold" which is severe for all users. In other words, the database contains severe vulnerabilities for software and operating systems that many users interact with on a daily basis. One is able to view the vulnerabilities within seven predetermined metrics: Name, ID Number, CVE name, Date Public, Date Published, Date Updated, and Severity Metric. Even though US-CERT does not offer any classification, the search feature is very good. It does a full text search allowing the user to input complex queries. Figure 1.5 shows a snapshot of the US-CERT vulnerability notes search page. Figure 1.6 shows a snapshot of the results returned when searching for buffer overflow 2006. The US-CERT Vulnerability Notes Database is a truncated list of CVE vulnerabilities. Therefore, one is only able to get a subset of CVE vulnerabilities as a result. Another downfall comes as the results from the search as featured in figure 5 contain vulnerabilities from years other than 2006 with no indication of the number of vulnerabilities that have been returned. The user must do extra work to know how many vulnerabilities fit into the search results. In addition, extra time must be sent filtering through the results to find the ones that are from 2006.

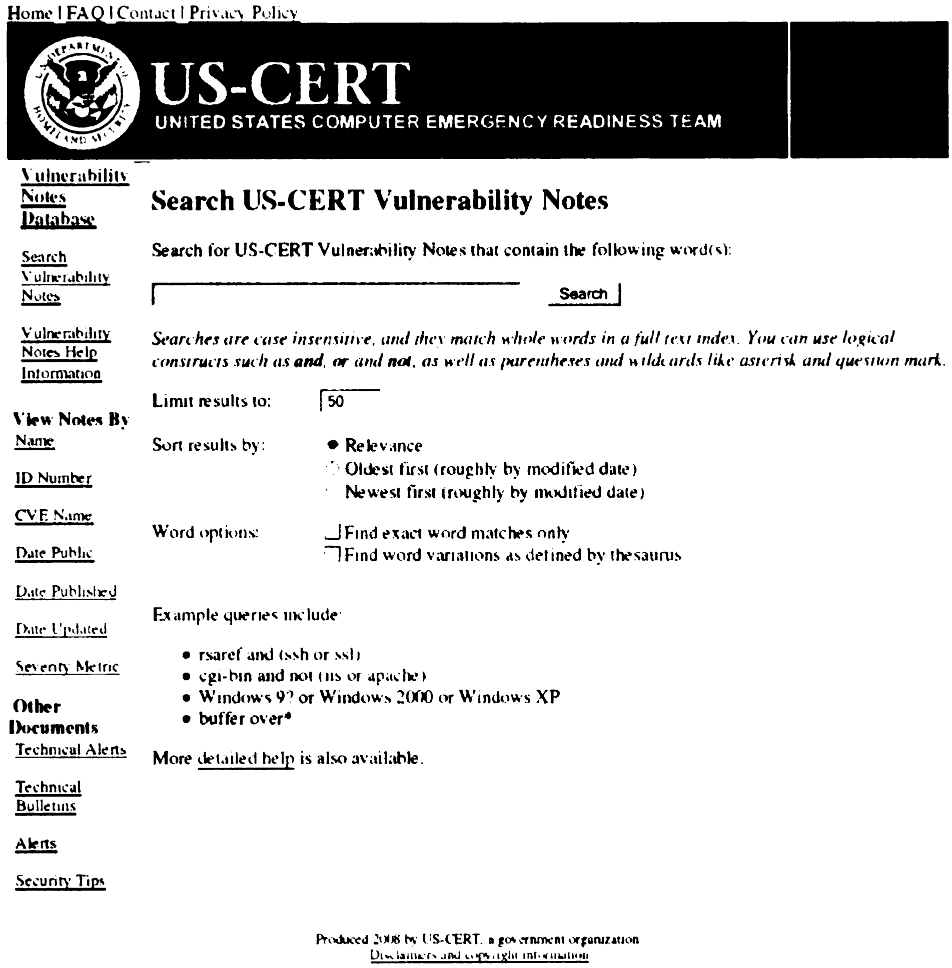


Figure 1.5. A screenshot of US-CERT vulnerability search feature

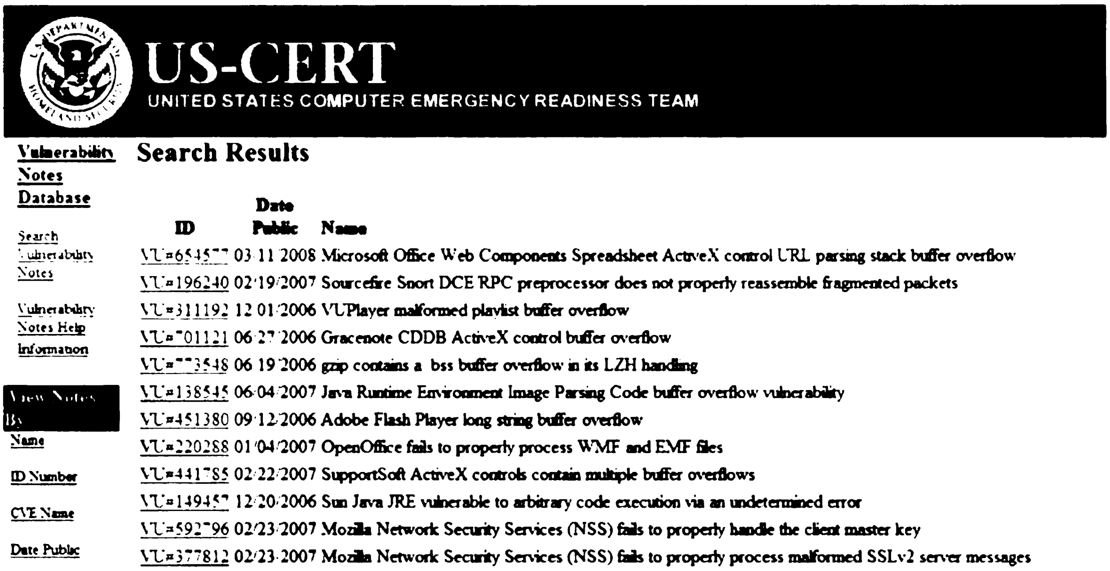


Figure 1.6. A screenshot of the US-CERT search results for buffer and overflow and 2006

The National Vulnerabilities Database, NVD, which is sponsored by the National Institute of Science and Technology, is a very interesting database to study. It is a CVE and CCE vulnerability database. Table 1.3 shows which CWE vulnerability classifications are integrated into NVD. Table 1.3 is mainly composed of CWE tier 2 vulnerability classifications. It is interesting to note that because the list is a truncated version, classifications such as Other and Not in CWE exist. The user is able to do a keyword search with any of the one of the CWE metrics from table 2 specified, any of the metrics within CVSS specified, the product specified, the vendor specified, or the date specified. Figure 1.7 helps to illustrate the search and browsing features of NVD.

Name	Description
Authentication Issues	Failure to properly authenticate users.
Credentials Management	Failure to properly create, store, transmit, or protect passwords and other credentials.
Permissions, Privileges, and Access Control	Failure to enforce permissions or other access restrictions for resources, or a privilege management problem.
Buffer Errors	Buffer overflows and other buffer boundary errors in which a program attempts to put more data in a buffer than the buffer can hold, or when a program attempts to put data in a memory area outside of the boundaries of the buffer.
Cross-Site Request Forgery (CSRF)	Failure to verify that the sender of a web request actually intended to do so. CSRF attacks can be launched by sending a formatted request to a victim, then tricking the victim into loading the request (often automatically), which makes it appear that the request came from the victim. CSRF is often associated with XSS, but it is a distinct issue.
Cross-Site Scripting (XSS)	Failure of a site to validate, filter, or encode user input before returning it to another user's web client.
Cryptographic Issues	An insecure algorithm or the inappropriate use of one; an incorrect implementation of an algorithm that reduces security; the lack of encryption (plaintext); also, weak key or certificate management, key disclosure, random number generator problems.
Path Traversal	When user-supplied input can contain ".." or similar characters that are passed through to file access APIs, causing access to files outside of an intended subdirectory.
Code Injection	Causing a system to read an attacker-controlled file and execute arbitrary code within that file. Includes PHP remote file inclusion, uploading of files with executable extensions, insertion of code into executable files, and others.
Format String Vulnerability	The use of attacker-controlled input as the format string parameter in certain functions.
Configuration	A general configuration problem that is not associated with passwords or permissions.
Information Leak / Disclosure	Exposure of system information, sensitive or private information, fingerprinting, etc.
Input Validation	Failure to ensure that input contains well-formed, valid data that conforms to the application's specifications. Note: this overlaps other categories like XSS, Numeric Errors, and SQL Injection.
Numeric Errors	Integer overflow, signedness, truncation, underflow, and other errors that can occur when handling numbers.
OS Command Injections	Allowing user-controlled input to be injected into command lines that are created to invoke other programs, using system() or similar functions.

Table 1.3. CWE classifications used by NVD

Name	Description
Race Conditions	The state of a resource can change between the time the resource is checked to when it is accessed.
Resource Management Errors	The software allows attackers to consume excess resources, such as memory exhaustion from memory leaks, CPU consumption from infinite loops, disk space consumption, etc.
SQL Injection	When user input can be embedded into SQL statements without proper filtering or quoting, leading to modification of query logic or execution of SQL commands.
Link Following	Failure to protect against the use of symbolic or hard links that can point to files that are not intended to be accessed by the application.
Other	NVD is only using a subset of CWE for mapping instead of the entire CWE, and the weakness type is not covered by that subset.
Not in CWE	The weakness type is not covered in the version of CWE that was used for mapping.
Insufficient Information	There is insufficient information about the issue to classify it; details are unkown or unspecified.
Design Error	A vulnerability is characterized as a "Design error" if there exists no errors in the implementation or configuration of a system, but the initial design causes a vulnerability to exist.

Table 1.3 continued

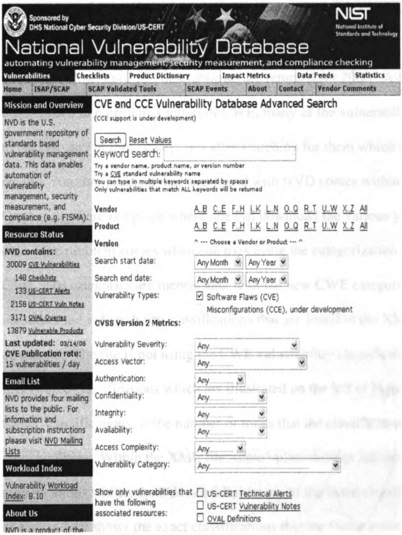


Figure 1.7. A screenshot of NVD vulnerability search

Despite the positive features of NVD, there are some other features which cause confusion. The search engine provides many search options while the key word search is not very extensive. In fact, when searching for buffers overflows, there were some vulnerabilities that were identified as buffer overflow vulnerabilities that were not being returned. Figure 1.8 illustrates the search results returned when typing buffer overflow into the search with the year of 2006 specified. There are 583 vulnerabilities that are returned. However, if one selects just buffer errors from 2006 as specified within Figure 1.9, only 30 vulnerabilities are returned. When limiting the buffer errors to the keywords buffer overflow, only 19 vulnerabilities are returned for 2006. As 23 classifications have been chosen to represent a subset of CWE, many of the vulnerabilities have not been set to fall into any of these categories after searching for them which leads to inaccurate results. Another problem that was found with NVD comes within the XML downloads. On the site, there is a place where one can download the various years' worth of CVE data. The problem comes when one tries using the categorization aspects within the CVE. The categories are inconsistent with the new CWE categories. Figure 1.10 illustrates the vulnerability classifications that are found in the XML of the 2007 download. The file is not using the CWE vulnerability classification standard. Instead it lists twelve classifications which are illustrated on the left of Figure 1.10. Next to the twelve classifications, is the number of times that the classification is found within the vulnerabilities. Within the XML file, some vulnerabilities fall into several of the classifications while others do not fall into any of the listed classifications. The right side of Figure 1.10 shows the exact classifications that are found associated each vulnerability is associated with, as well as the number of times that classification is found within the vulnerabilities. According to Bishop, allowing multiple categories can be a good thing so this may prove to be a problem. However, NVD is using two different vulnerability classification metrics. Another concern is that 542 of the vulnerabilities are classified as unknown.

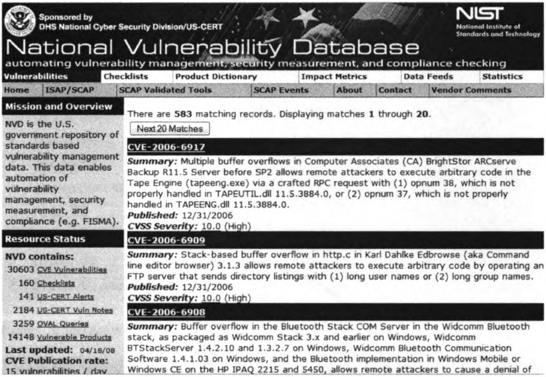


Figure 1.8. A screenshot of the search results returned from NVD when buffer overflow is typed into the keyword search and the year 2006 is specified

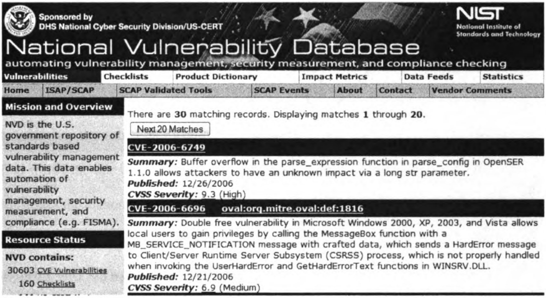


Figure 1.9. A screenshot of the search results returned from NVD when the CWE category of buffer errors is selected from the list of vulnerability classifications.

- input buffer 614
- input bound 74
- unknown 542
- env 25
- design 952
- input 4352
- config 68
- exception 361
- race 39
- access 305
- other 44
- input bound buffer 2

- input buffer 602
- input bound 67
- unknown 542
- env 15
- design 723
- input 4107
- config 37
- exception 271
- design, race 4
- race 34
- input bound, exception 1
- access 202
- input, design 128
- Input, exception 49
- design, env 3
- design, exception 28
- access, design 49
- access, input 38
- access, config 3

- other 42
- Input bound, design 5
- input bound buffer 2
- design, config 6
- input, config 20
- input bound, other 1
- input buffer, other 1,
- access, input buffer 3,
- input buffer, design 3
- input, race 1
- input, env 5
- input, exception, env 1
- input buffer, exception 3
- exception, env 1
- input buffer, config 2
- access, input, design 3
- access, exception 7

Figure 1.10. Illustration of NVD vulnerability categorization confusion

Open Source Vulnerability Database,OSVDB, offers an extensive vulnerability database which incorporates: Bugtraq ID, CVE ID, ISS X-Force ID, Nessus Script ID, Related OSVDB ID, Snort Signature ID, Secunia Advisory ID, FrSIRT Advisory ID, OVAL ID, CIAC Advisory, CERT, CERT VU, Security Tracker, and Milw0rm ID. OSVDB has made several vulnerability classifications which include Location, Attack Type, Impact, Solution, Exploit, Disclosure, OSVDB. The search feature allows the user to select any the vulnerability classifications, select a reference point, input key words to find within the title or text, the vendor or product name, or a time period. Figure 1.11 illustrates the elaborate search capabilities. The limitation of OSVDB comes within the search. Keywords are only allowed to be found within the vulnerability titles. By allowing only keywords to be searched for within the title, one is not able to get access to all vulnerabilies. OSVDB lists it's vulnerability classification within the title. If one picks search terms that are too general or specific for the title, they will not receive the proper results. Figure 1.12 contains the three search results that are returned for buffer overflow for the year of 2006. It is hard to believe that only three of the vulnerabilities from 2006 were buffer overflows.

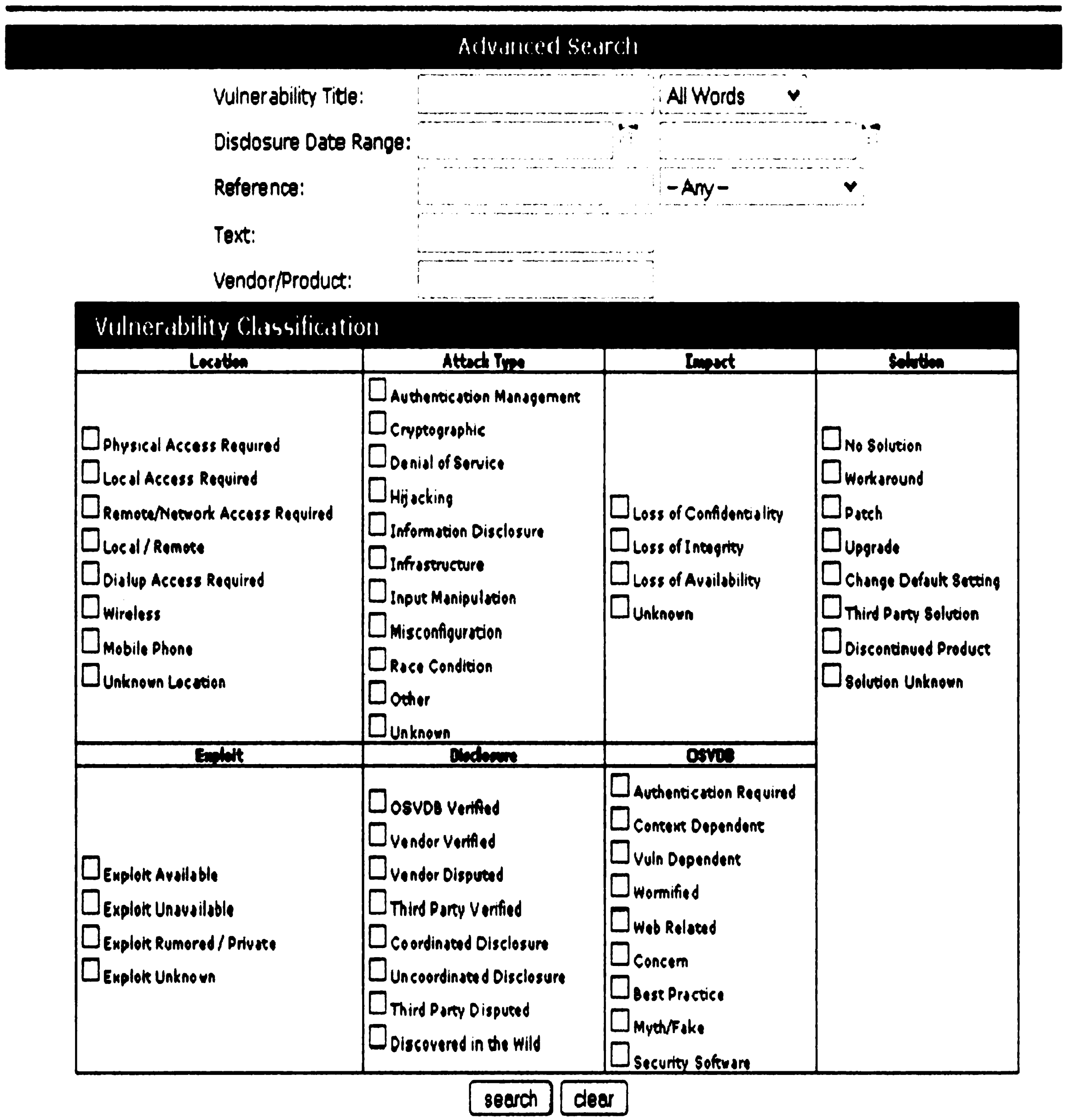


Figure 1.11. OSVDB vulnerability search

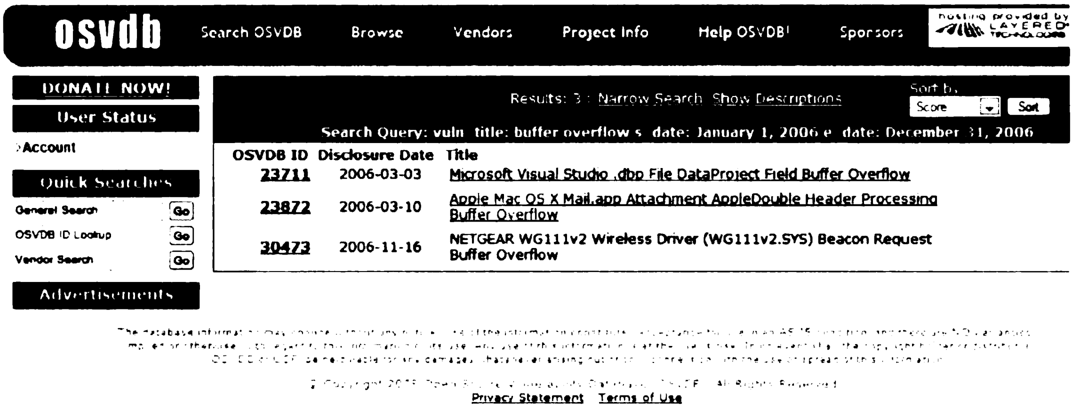


Figure 1.12. The search results returned when searching for buffer overflows for the year 2006

Secunia offers an advisory, vulnerability, and virus database. Browsing options include historic advisories, listed by product, and listed by vendor. Categorization within Secunia includes Impact, Critical Levels, and Where. Users are allowed to input key words as well as select an option from the different categorizations when searching through the database. Figure 1.13 shows a snapshot of the advanced search part of the website. Figure 1.14 shows the results returned when searching for buffer overflow 2006. 2006 was added into the search to find vulnerabilities only from the year 2006. However, vulnerabilities from 2007 are featured on within the results in figure 1.14. Overall, Secunia offers a great search tool combined with basic classifications.

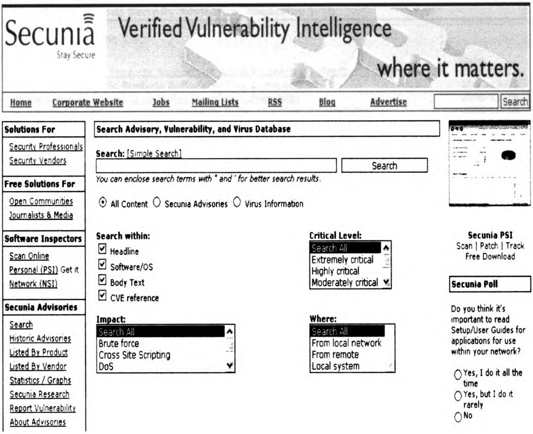


Figure 1.13. Secunia vulnerability search

Found: 483 Secunia Security Advisories, displaying 1-25

Sort by: Match, Title, Date

Title	Date
Symantec Multiple Products SupportSoft ActiveX Controls Buffer Overflow	2007-02-23
JustSystems Multiple Products Buffer Overflow Vulnerability	2006-12-05
Borland Products idsql32.dll Buffer Overflow Vulnerability	2006-11-29
JustSystems Ichitaro Document Property Buffer Overflow Vulnerability	2006-10-18
Ipswitch IMail Server SMTP Service Buffer Overflow Vulnerability	2006-09-07
Ichitaro Document Viewer Buffer Overflow Vulnerability	2006-08-21
Microsoft Visual Basic for Applications Buffer Overflow	2006-08-08
PowerArchiver DZIPS32.DLL Buffer Overflow Vulnerability	2006-07-25
Microsoft Office Image Filters Buffer Overflow Vulnerabilities	2006-07-11
Microsoft Excel Multiple Buffer Overflow Vulnerabilities	2006-07-06
McAfee SecurityCenter Subscription Manager Buffer Overflow	2006-08-01
Alien Arena 2006 Gold Edition Multiple Vulnerabilities	2006-03-08
Symantec Support Tool ActiveX Control Vulnerabilities	2006-10-06
Microsoft Word Code Execution Vulnerabilities	2006-09-05
Mandriva update for xine-lib	2006-06-26
Microsoft Office Multiple Code Execution Vulnerabilities	2006-03-14
F-Secure Anti-Virus Archive Handling Vulnerabilities	2006-01-19
Graphviz GD GIF Handling Buffer Overflow Vulnerability	2008-02-13
Visual Studio Crystal Reports RPT Processing Buffer Overflow	2007-09-11
Media Player Classic FLI File Processing Buffer Overflow	2007-08-24
Microsoft DirectX RLE Compressed Targa Image Processing Buffer Overflow	2007-07-19
Avaya Products GDB "DWARF" Buffer Overflow Vulnerabilities	2007-07-04
Cisco Products PHP "htmlentities()" and "htmlspecialchars()" Buffer Overflows	2007-04-26
IBM Lotus Domino Script Insertion and Buffer Overflows	2007-03-28
Gentoo mgv Buffer Overflow Vulnerability	2007-03-27

Next 25 matches >>

Figure 1.14. A screenshot of results from buffer overflow2006 as the key word search

1.4 Trend analysis

NVD is the only vulnerability database with any sort of trend analysis. Figure 1.15 shows the trend selections that one can make. The image shows that the capabilities are basically the same as the search, but there is no key word search. Therefore, the user is not able to gain any trend analysis from customized search results. Figure 1.16 demonstrates the statistical results while Figure 1.17 shows the graphs of the results from choosing buffer errors from the Vulnerability Category and selecting the time period of January 2003 through March 2008. Figure 1.16 indicates that the statistics are not being calculated correctly. Within the statistics given, it claims that there are 29 buffer error vulnerabilities in 2006 which constitutes to 0% of the makeup of vulnerabilities. It is not possible for 29 vulnerabilities to constitute for 0% of the vulnerabilities in 2006. The error is due to the high number of vulnerabilities in 2006. There should still be some way to computer the actual value or provide a more accurate computation.

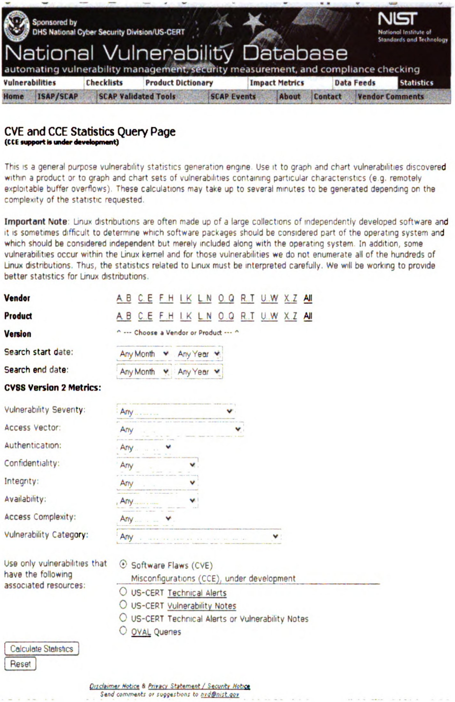
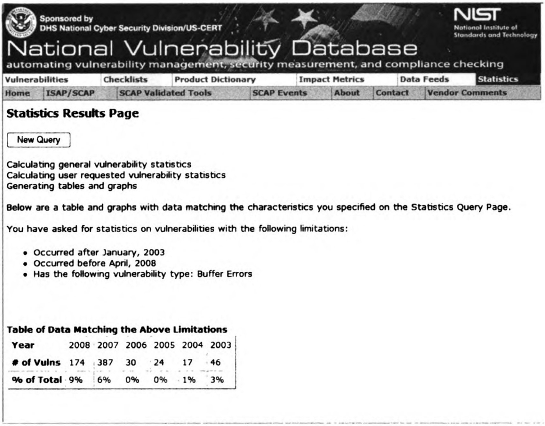


Figure 1.15. Criteria selection when using the NVD statistics query page


Sponsored by
DHS National Cyber Security Division/US-CERT

NIST
National Institute of Standards and Technology

National Vulnerability Database
automating vulnerability management, security measurement, and compliance checking

Vulnerabilities | Checklists | Product Dictionary | Impact Metrics | Data Feeds | Statistics
Home | ISAP/SCAP | SCAP Validated Tools | SCAP Events | About | Contact | Vendor Comments

Statistics Results Page

New Query

Calculating general vulnerability statistics
Calculating user requested vulnerability statistics
Generating tables and graphs

Below are a table and graphs with data matching the characteristics you specified on the Statistics Query Page.

You have asked for statistics on vulnerabilities with the following limitations:

- Occurred after January, 2003
- Occurred before April, 2008
- Has the following vulnerability type: Buffer Errors

Table of Data Matching the Above Limitations

Year	2008	2007	2006	2005	2004	2003
# of Vulns	174	387	30	24	17	46
% of Total	9%	6%	0%	0%	1%	3%

Figure 1.16. The statistical results of choosing buffer errors from the Vulnerability Category and selecting the time period of January 2003 through March 2008

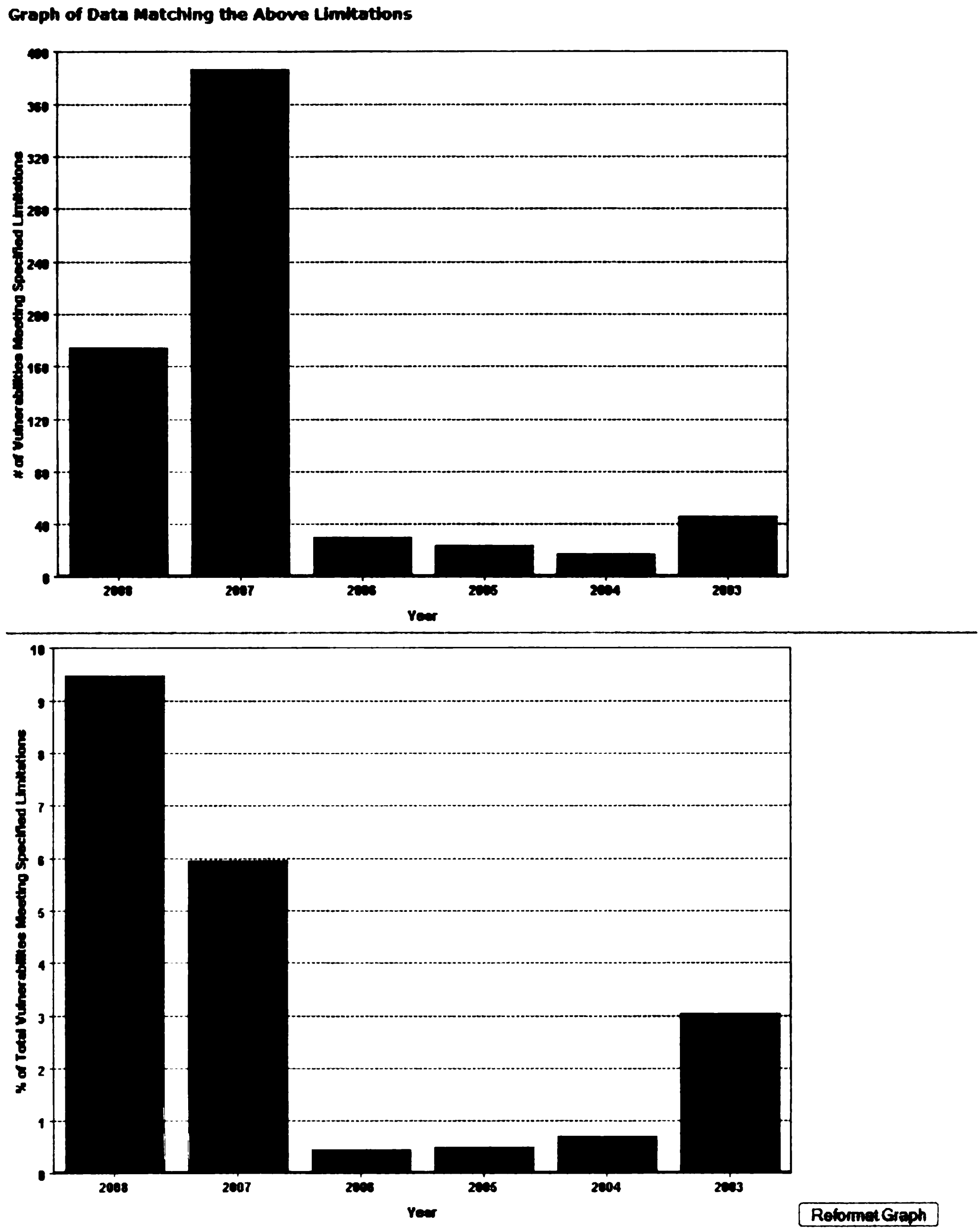


Figure 1.17. The graphical results of choosing buffer errors from the Vulnerability Category and selecting the time period of January 2003 through March 2008

CWE also offers a form of trend analysis.(Christey) Because CWE is not a vulnerability database, they only offer the statistics of CVE vulnerabilities broken down into 41 classifications. The trend analysis is more of a statistical study which offers only tables. A sample of the analysis is shown by Figure 1.18. The rows are ordered by the classifications used while the columns indicate the totals from 2001 until 2006. Anyone that uses the provided trend analysis has no concept of which vulnerabilities fall into the classifications used in the analysis. Therefore, the user would have a hard time making any changes to the classifications. One is also able to find the importance of a graph after reading from the large tables as it is hard to put the tables into perspective.

Table 1: Overall Results

Rank	Flaw	TOTAL	2001	2002	2003	2004	2005	2006
Total		**18809**	**1432**	**2138**	**1190**	**2546**	**4559**	**6944**
[1]	**xss**	**13.8%**	[illegible]	08.7% (2)	07.5% (2)	10.9% (2)	16.0% (1)	18.5% (1)
		2595	*31*	*187*	*89*	*278*	*728*	*1282*
[2]	**buf**	**12.6%**	19.5% (1)	20.4% (1)	22.5% (1)	15.4% (1)	09.8% (3)	07.8% (4)
		2361	*279*	*436*	*268*	*392*	*445*	*541*
[3]	**sql-inject**	**09.3%**	[illegible]	[illegible]	03.0% (4)	05.6% (3)	12.9% (2)	13.6% (2)
		1754	*6*	*38*	*36*	*142*	*588*	*944*
[4]	**php-include**	**05.7%**	[illegible]	[illegible]	[illegible]	[illegible]	02.1% (6)	13.1% (3)
		1065	*1*	*7*	*12*	*36*	*96*	*913*
[5]	**dot**	**04.7%**	08.9% (2)	05.1% (4)	02.9% (5)	04.2% (4)	04.3% (4)	04.5% (5)
		888	*127*	*110*	*34*	*106*	*196*	*315*
[6]	**infoleak**	**03.4%**	02.6% (9)	04.2% (5)	02.8% (6)	03.8% (5)	03.8% (5)	03.1% (6)
		646	*37*	*89*	*33*	*98*	*175*	*214*
[7]	**dos-malform**	**02.8%**	04.8% (3)	05.2% (3)	02.5% (8)	03.4% (6)	01.8% (8)	02.0% (7)
		521	*69*	*111*	*30*	*86*	*83*	*142*
[8]	**link**	**01.8%**	04.5% (4)	02.1% (9)	03.5% (3)	02.8% (7)	01.9% (7)	[illegible]
		341	*64*	*45*	*42*	*72*	*87*	*31*
[9]	**format-string**	**01.7%**	03.2% (7)	01.8% (10)	02.7% (7)	02.4% (8)	01.7% (9)	00.9% (11)
		317	*46*	*39*	*32*	*62*	*76*	*62*
[10]	**crypt**	**01.5%**	03.8% (5)	02.7% (6)	01.5% (9)	[illegible]	01.5% (10)	00.8% (13)
		278	*55*	*58*	*18*	*22*	*69*	*56*
[11]	**priv**	**01.3%**	02.5% (10)	02.2% (8)	01.1% (12)	01.3% (11)	01.5% (11)	00.8% (14)
		249	*36*	*46*	*13*	*33*	*67*	*54*
[12]	**perm**	**01.3%**	02.7% (8)	01.8% (11)	01.3% (11)	00.9% (15)	01.1% (13)	01.1% (9)
		241	*39*	*39*	*15*	*24*	*48*	*76*

Figure 1.18. A section of the CWE trend analysis

OSVDB has announced plans for a statistics project to for Google Summer of Code 2008.

> This project is to create a flexible framework that can provide useful statistics on vulnerabilities from OSVDB. This project should take in consideration all of the fields and classifications in OSVDB.
> -Should create and generate standard/most popular graphs and charts each day and make available
> -Should create statistics that allows very flexible/detailed stats to be dynamically generated on demand by user
> -Some examples of statistics required:
> -# Vulns based on Disclosure Year
> -Detailed stats based on each vuln classification options (ALL OPTIONS)
> -# of vulns by Vendor
> -# of vulns by Product
> -# of vulns that do not have a solution (and by vendor)
> -Time from when a vuln was discovered and then disclosed
> -Create stats application that allows user to dynamically generate stats based on their own requirements.
> -Trend the number of vulns released per day ("OSVDB GSoC 2008 Project Ideas.")

Although this has not yet been implemented, it helps to illustrate the need for vulnerability analysis. From the list of features that are provided, VACT will be able to accomplish all tasks except for finding the time from when a vulnerability was discovered and then disclosed and the number of vulnerabilities released per day. The user will be able to use the options within VACT to accomplish all other trends.

Below is a summarization of the problems that were found within the vulnerability databases.

- Each vulnerability database only allows one search at a time
- There is no way to obtain a copy of the search results
- Each of the vulnerability databases returned a different number of results when searching for buffer overflow within the year 2006.
- US-CERT contains a truncated list of vulnerabilities

- NVD does not follow the vulnerability classifications
- OSVDB only searches through the vulnerability title
- US-CERT, Secunia, and OSVDB do not offer trend analysis
- NVD trend analysis is not customizable

Chapter 2:VACT Overview

2.1 Framework

Customizability is an important characteristic of this project which sets it apart from other tools. One way to provide customization is to use a powerful yet simple programming language that can allow the user to make some changes without the need to develop a tool to for the task that is being accomplished. VACT requires no programming knowledge to work, however the knowledge of programming allows one to customize the tool. Another way to aid users is to provide a flexible tool that has enough range to supply useful information to a variety of users.

The purpose of VACT is to provide a quick and easy way to search through and analyze vulnerabilities. VACT brings together vulnerability classification, vulnerability search, and trend analysis. In order for this to happen, the tool needs to provide a common interface so that the user does not have to search through various vulnerability databases for vulnerabilities. For the interface to be successful, it must be able to accommodate the needs of many different users. To make this possible we rely on a simple, robust front end solution.

In order to provide a solid framework, Python is the chosen programming language. It offers a powerful yet robust object-oriented environment. VACT is set up so that it must gather vulnerabilities from vulnerability databases and then parse through a large amount of text. Python does especially well with string processing and Internet retrieval.

2.2 Classification and Search

We found a lot of disparity within the classification schema of vulnerability databases. Even within all the various classification options, we were not able to select the vulnerabilities out of the databases that we wanted. We could not find a vulnerability database that could find buffer errors within the past year that allowed privilege elevations. Therefore, a main feature within the Vulnerability Analysis and Classification Tool is to allow the user to specify the classifications. Our approach allows the user to preselect all the keywords that are necessary and then perform a search. Another feature for user convenience, which no vulnerability databases have the ability to do is to allow the user to perform multiple vulnerability searches for comparison. Figure 2.1 shows of screenshot of Vulnerability Analysis and Classification Tool. Within the figure, one is able to see the simplicity of the design. Section 3.2 explains the user interface in detail.

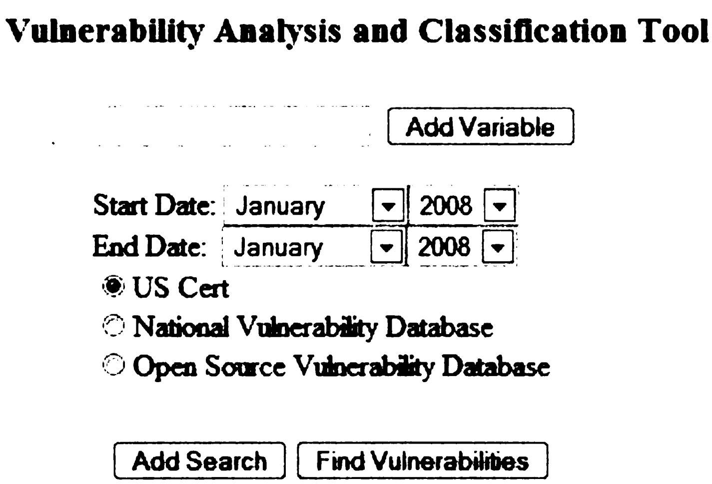


Figure 2.1. A Screenshot of VACT when first initialized

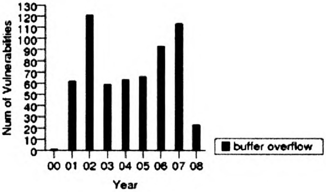
Allowing the user to specify the classification creates a better tool than if it were using preclassified vulnerabilities. Even though CWE has so many great features, it is constantly changing. From December 2007 to April of 2008, CWE has gone from draft 7 to draft 9. As the drafts changed so have the various classification schemas that are used within the drafts. The classifications given do not have all the possible vulnerabilities listed with them, therefore, one would have to classify each vulnerability according to the classification tree within CWE that is being used. Another limitation to using CWE is that when a new classification is made, the classification is not instantly changed within the table. Allowing users to search for vulnerabilities by making their own classifications alleviates the problems stated above as the user now has the freedom to use CWE as a reference or make up their own classifications.

2.3 Trend Analysis

After the vulnerabilities are gathered, VACT returns the results to the user in graphical form. Figure 2.2 illustrates an example of an output page returned to a user. The user is also given an option to download a CSV file containing the vulnerabilities found to match the classification criteria. A simple set of statistics are also returned from the search. The statistics include the total number of vulnerabilities, the total number of vulnerabilities within each category, and a breakdown of which characteristics are found within the vulnerabilities. There are currently no vulnerability databases which allow a user to download the vulnerability search results or that returns a user trend analysis information based upon the vulnerability search results. Section 4.1 will explain the trend analysis functionality in farther detail.

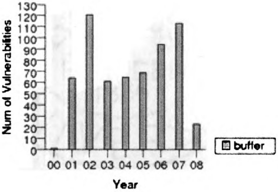
Results

Analysis 1



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	300	374	257	339	284	416	366	67	2404
Matching	1	62	121	59	63	66	93	113	23	601
Percent	3.2	20.7	32.4	23.0	18.6	23.2	22.4	30.9	34.3	25.0

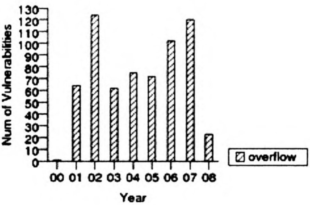
Analysis 2



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	300	374	257	339	284	416	366	67	2404
Matching	1	64	121	61	65	69	94	113	23	611
Percent	3.2	21.3	32.4	23.7	19.2	23.2	22.6	30.9	34.3	25.4

Figure 2.2. Results page from VACT

Analysis 3



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	300	374	257	339	284	416	366	67	2404
Matching	1	64	124	62	75	72	102	120	23	643
Percent	3.2	21.3	33.2	24.1	22.1	25.4	25.0	32.8	34.3	26.7

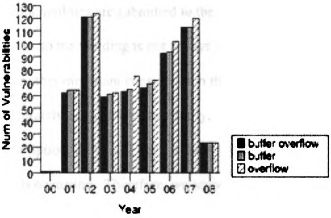


Figure 2.2 continued

Chapter 3: Classification and Search

3.1 Strategy

As one has gathered from the background, two key components of Vulnerability Analysis and Classification Tool are the classification and search feature. The implementation of these tasks consist of three sections. The first task is gathering the user's vulnerability classification schema that will be searched. The next task is a behind the scenes act of gathering the database information. The third and final task is performing the search upon the gathered data.

Before going into detail of how the search is configured, I will first talk about the search strategy that is being used. String search was chosen on the basis that it offers a simple yet effective search and the vulnerability descriptions offer enough detail to make it happen. When making this choice there were a few tradeoffs that needed to be considered. Vulnerabilities are submitted to the various databases by various researchers and organizations, so the wording is not always consistent between vulnerability descriptions. Another important consideration that we take into account is that the descriptions accurately depict the vulnerability. Before someone submits a vulnerability and offers a description, they must have enough knowledge of the situation to know where the error is occurring. Therefore, we argue that if one has enough knowledge to find and submit a vulnerability, then the person is able to accurately describe where the application is failing.

3.2 User Interface

Vulnerability Analysis and Classification Tool uses a web-based interface to interact with the user. It is composed of an html form which relies on JavaScript to help add words and new search boxes. As seen in figure 3.1, the user interface allows for the user to input classification variables for a search, add another search, and submit the search queries. For any individual search, the user is allowed to specify the vulnerability database, the time frame to search, and any classification words or phrases that should be used within the search. To add a word or phrase, the user must input it into the associated textbox and select Add Word. The Add Word button calls a JavaScript function which adds the word or phrase into the html form. Finally the user is allowed to submit the search. Clicking the Submit button sends the associated form of search variables to the python search and analysis code.

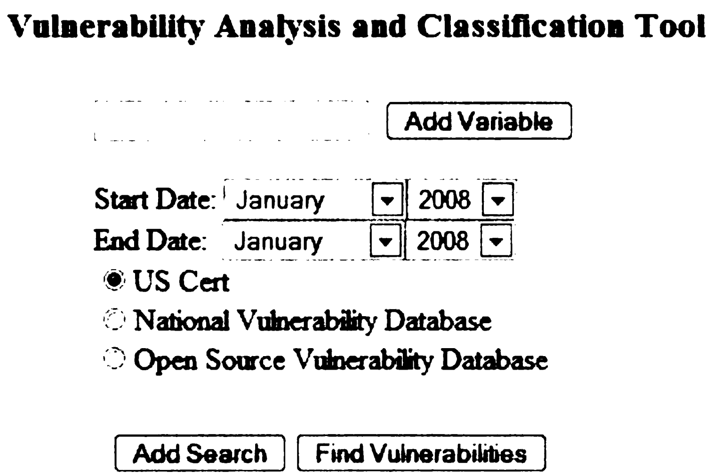


Figure 3.1 A screenshot of VACT when first initialized

3.3 Functionality

Upon reception of the search form, the database variables must be obtained. There are two main methods to obtain the vulnerability information from the vulnerability databases are crawling through the web site to gather the information that is necessary and downloading a premade file meant for download. Crawling through the site is necessary as some sites do not allow one to download the information that is available. Once the vulnerabilities are downloaded, no matter what the source, they are put into a Python dictionary. The dictionary uses the vulnerability name followed by the date as a key and a tuple containing the vulnerability description as a set paired with the date for the value.

The search function accepts the dictionary of vulnerabilities as well as the list of the classification words. The function loops through each vulnerability checking for each classification term in the list. If the classification is not found within the vulnerability description, then it is removed from the dictionary of vulnerabilies. Once the search has completed, we send the results to the trend analysis.

3.4 Efficiency

The runtime of VACT is dominated by the amount of data that must be downloaded. To illustrate the runtime of the search function, several test scenarios have been setup. The first test set uses the US-CERT vulnerabilities, which is composed of nearly 2,400 vulnerabilities. The vulnerabilities must be downloaded using the web crawler method. The second test set contains the 2007 NVD vulnerabilities which contains over 6,000 vulnerabilities. The third test set contains the NVD vulnerabilities from 2003-2007. The third test set contains over 22,000 vulnerabilities.

Downloading the first set of vulnerabilities, took an average time to be close to 1 minute and 30 seconds. While downloading the second set, from NVD took an average of 1 minute and 45 seconds. The third set took the longest at 5 minutes to download. Therefore, we recommend a preconfigured download if at all possible. The composition of the dictionary from each set of vulnerabilities took an average of one second to fill. Using the same vulnerabilities within the setup above, tests were run to see how long it would take to return the search results after the vulnerabilities are gathered. In the test, various trials were done by varying the classifications from 1-6 words and the number of searches from 1-6 searches. For example, one test contained four searches consisting of (buffer and overflow), (buffer), (overflow), and (exception, microsoft, the, a, and, .dll). The search for the example just like all other searches returned the results of all searches within two seconds for each trial set of vulnerabilities. Another test was run on the third set to make sure that there would be no problems searching the data. The third set was set to run with twelve search words. The search was returned within one second.

3.5 Naïve Bayesian Classification

String search is a fast and efficient way to match key words, yet there is a limitation to the string search. There are some vulnerability classifications which cannot be summarized within a reasonable amount of key words. To help with any such cases, Vulnerability Analysis and Classification Tool includes a generic naïve Bayesian approach. Naïve Bayesian is a classification algorithm that uses the probability of occurrence to classify an attribute. The naive Bayesian classification will take a csv that is returned from the string search as an input. The descriptions of each classification are combined into a dictionary with the words as the key and the number of occurances as the value. Once the dictionary is filled, the common words are stripped out. The Algorithm is now ready to find words that only occur once and only in one class. When a single word is found, it is tagged as unknown. The unknown words are added together as a special case that will handle words that do not belong into each class. We now have the words all sorted and categorized according to classification, the next step is to compute the probabilities of each class. P(c) is computed for each class by taking the number of occurrences of each class and dividing it by the sum of class occurrences. P(word | c) is computed for each word within the class by taking the occurrence of the word and dividing it by the total words in each class. Now that the probability of a classification and probability of a word within a classification are computed, the algorithm is ready to classify vulnerabilities. Figure 3.2 shows pseudo code used to compute the probability of a vulnerability description belongs to a classification. After computing the probability of each classification, the best one is returned and the vulnerability is associated within that classification.

```
For each class:
Pclass = P(c)
For words in sentence:
Pclass = Pclass * P(word | c)
```

Figure 3.2. Psuedo code showing how a vulnerability's probability is computed according to the naïve Bayesian classification

Naïve Bayesian classification is offered as a classification aide to help enhance the string search. Naïve Bayesian, as any other approaches, has its limitations. The largest limitation is that it is confined to the classifications that are given to the algorithm. If only two classifactions are given, any vulnerability must fall into one of the two possible classifications. When running the algorithm after computing the probabilities of the classifications, the user may find that the probability of a vulnerability does not fall into its original classification. Despite the limitations to naïve Bayesian classification, it presents users a way to search find new words and vulnerability classifications.

Chapter 4: Trend Analysis

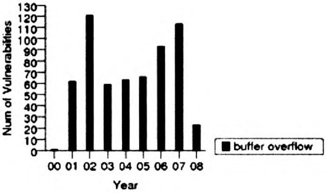
4.1 Evaluation of Features

The second key element within the thesis is the trend analysis as there is currently no implementation which encompasses the user's search results. The trends given are simple and efficient. They help to give the user a basic overview of how the classification schema presented fits in with the vulnerabilities. The trend analysis works by processing the results that are returned by the search and turning them into graphs, statistics, and a CSV file.

The results page for searching US-CERT for buffer overflow, buffer and overflow for the years 2000-2008 can be seen in Figure 4.1. Accompanying each search is a graph with the number of matching vulnerabilities. The matching vulnerabilities are the vulnerabilities that match the classification as defined within the search. The table associated with the graph shows the total number of vulnerabilities found per year, the number of matching vulnerabilities found per year, and the percent of vulnerabilities which are matching per year. If a user clicks on the words Analysis, then a CSV containing the matching vulnerabilities will become available. For example, the CSV for Analysis 1 would contain 601 entries. The CSV file contains the identification number of the vulnerability within the database searched, the date the vulnerability was published, and the description of the vulnerability.

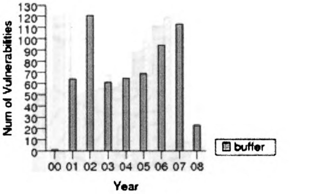
Results

Analysis 1



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	300	374	257	339	284	416	366	67	2404
Matching	1	62	121	59	63	66	93	113	23	601
Percent	3.2	20.7	32.4	23.0	18.6	23.2	22.4	30.9	34.3	25.0

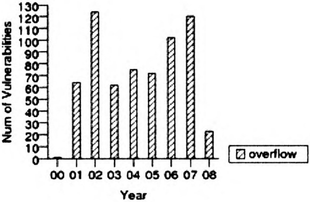
Analysis 2



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	300	374	257	339	284	416	366	67	2404
Matching	1	64	121	61	65	69	94	113	23	611
Percent	3.2	21.3	32.4	23.7	19.2	23.2	22.6	30.9	34.3	25.4

Figure 4.1. Results page from VACT

Analysis 3



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	300	374	257	339	284	416	366	67	2404
Matching	1	64	124	62	75	72	102	120	23	643
Percent	3.2	21.3	33.2	24.1	22.1	25.4	25.0	32.8	34.3	26.7

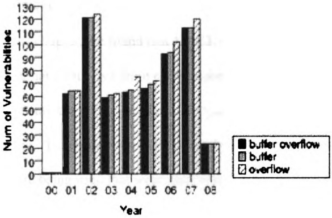


Figure 4.1 continued

The graphs are computed with the aid of PyChart. (Saito) PyChart is a graphing utility written by Yasushi Saito. VACT contains a function linking the search results to the graphing utility. Once the images are created, they are displayed within the results page. Referring back to Figure 4.1, one can see that the graphs given are a breakdown of the vulnerabilities over time. The bar chart plots are returned per vulnerability classification searched one in terms of years. The results from each search are also put into a large bar chart.

The CSV file is made by writing the categories used within the search to file. The basic format includes vulnerability id, vulnerability date, and vulnerability description. Because the dictionary contains the key of vulnerability name, vulnerability date with the value of the vulnerability description, it only takes one loop through each of the values to create the CSV. While looping through to create the CSV, the vulnerabilities are counted by their associated year. Therefore creating the graphs becomes easy as counting the values contained within each list. The set of statistics returned includes the breakdown of vulnerabilities per year as well as the total vulnerabilities found.

4.2 Efficiency

In the last chapter, we found that VACT is limited by the time it takes to download the vulnerability lists from each database being searched. Even with the download time being the dominate factor, we would like the trend analysis to be efficient like the searching. To perform the trend analysis test setup, the same sets of vulnerabilities are being used that were used within the search (USCERT vulnerabilities, NVD 2007 vulnerabilities, and NVD 2003-2007 vulnerabilities). Each set of vulnerabilities was used as a result set that was passed to the trend analysis. Generating the statistics and writing to the CSV file, took less than 20 seconds. Creating graphs for each of the sets took an average time of 5 seconds.

Chapter 5: Real World Example

5.1 Problem

Every now and again there is an article stating which operating system provides the best security. Many times the writer of the article bases the fact on the number of possible vulnerabilities that exist within the various operating systems. At one point, the author based the security on how fast that vulnerabilities have been patched. One is able to use Vulnerability Analysis and Classification Tool to verify that the user is giving accurate results.

5.2 Results

When searching for the results I was faced with some interesting problems. Should the results come from US CERT because they deal only with severe vulnerabilities, NVD because they offer all CVE vulnerabilities or OSVDB because it offers the most vulnerabilities. To show the flexibility and why authors might report various results, I decided to compute the results from each database. Another problem comes within the classification that was used to find the resulting set of vulnerabilities. The set of vulnerability classifications that were decided upon is illustrated within Tables 5.1 and 5.2. Both tables provide a summary for the results obtained using VACT. Table 5.1 is specific to the Microsoft classifications that are used while Table 2 is specific to the Apple search classifications that are used.

Figures 5.1-5.18 display the results returned by VACT. The data obtained from VACT illustrates a vast difference not only with the databases, but also within the classification that is used for the search. There is currently no quick and easy way to achieve the same results within the individual vulnerability databases. Within each vulnerability database, the search would need to be run a minimum of 13 different times using the various classifications indicated within Tables 5.1 and 5.2. After each search, the user would need to record the results returned to gather the statistics. The individual would then need to compute the trends. Even after going through this process, the user would not have lists of vulnerabilities found by each classification.

When viewing Table 5.1, one will find that Microsoft classifies 474 vulnerabilities within US-CERT, 852 Vulnerabilities within NVD and 771 vulnerabilities within OSVDB. One will also find this variance throughout the search results. To gain an accurate scope of the vulnerabilities, one would need to look through the CSV files to find the discrepancies within the vulnerabilities returned to obtain an overall count of vulnerabilities for both Microsoft and Apple.

NVD

Search String	Results	Percent of Total	VACT Figure
Microsoft	1116	4.08	5.8
microsoft, windows	432	1.58	5.8
microsoft, windows, xp	260	0.95	5.9
microsoft, windows, vista	46	0.17	5.9
microsoft, xp	552	2.02	5.10
microsoft, vista	46	0.17	5.10
windows xp	399	1.46	5.11
windows vista	55	0.20	5.11
Xp	1887	6.90	5.12
Vista	67	0.25	5.12

US-CERT

Search String	Results	Percent of Total	VACT Figure
Microsoft	422	19.19	5.1
microsoft, windows	131	5.96	5.1
microsoft, windows, xp	32	1.46	5.2
microsoft, windows, vista	2	0.09	5.2
microsoft, xp	131	5.96	5.3
microsoft, vista	2	0.09	5.3
windows xp	40	1.82	5.4
windows vista	6	0.27	5.4
Xp	319	17.74	5.5
Vista	6	0.27	5.5

OSVDB

Search String	Results	Percent of Total	VACT Figure
Microsoft	1128	2.71	5.15
microsoft, windows	376	0.90	5.15
microsoft, windows, xp	167	0.40	5.16
microsoft, windows, vista	40	0.10	5.16
microsoft, xp	490	1.18	5.17
microsoft, vista	40	0.10	5.17
windows xp	276	0.66	5.18
windows vista	51	0.12	5.18
Xp	2079	5.00	5.19
Vista	72	0.17	5.19

Table 5.1. Summary of results and classifications from VACT pertaining to Microsoft

NVD

Search String	Results	Percent of Total	VACT Figure
Apple	524	1.92	5.13
mac os x	442	2.62	5.13
apple, mac os x	264	.97	5.14

US-CERT

Search String	Results	Percent of Total	VACT Figure
Apple	139	6.32	5.6
mac os x	64	2.91	5.6
apple, mac os x	54	2.46	5.7

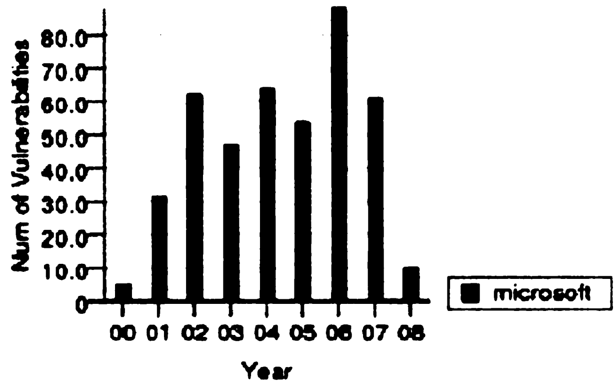
OSVDB

Search String	Results	Percent of Total	VACT Figure
Apple	351	0.84	5.20
mac os x	477	1.15	5.20
apple, mac os x	102	0.25	5.21

Table 5.2. Summary of results and classification from VACT pertaining to Apple

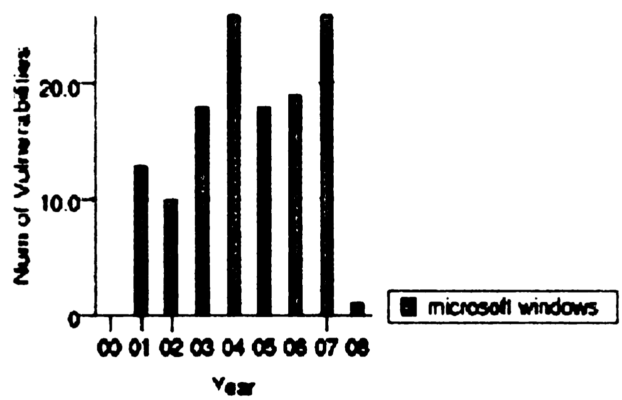
Results

Analysis 1



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	5	31	62	47	64	54	88	61	10	422
Percent	16.13	12.45	16.99	23.86	18.88	21.26	26.99	16.67	13.89	19.19

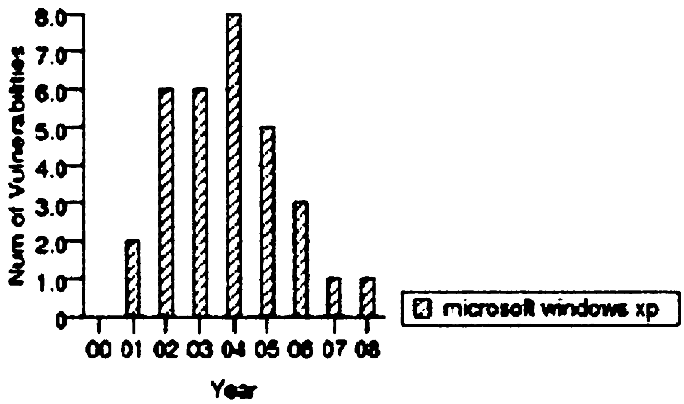
Analysis 2



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	13	10	18	26	18	19	26	1	131
Percent	0.00	5.22	2.74	9.14	7.67	7.09	5.83	7.10	1.39	5.96

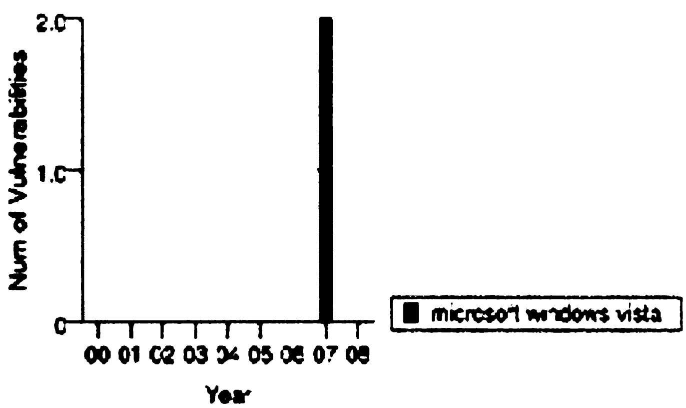
Figure 5.1. VACT results searching for Microsoft and Microsoft windows within US-CERT

Analysis 3



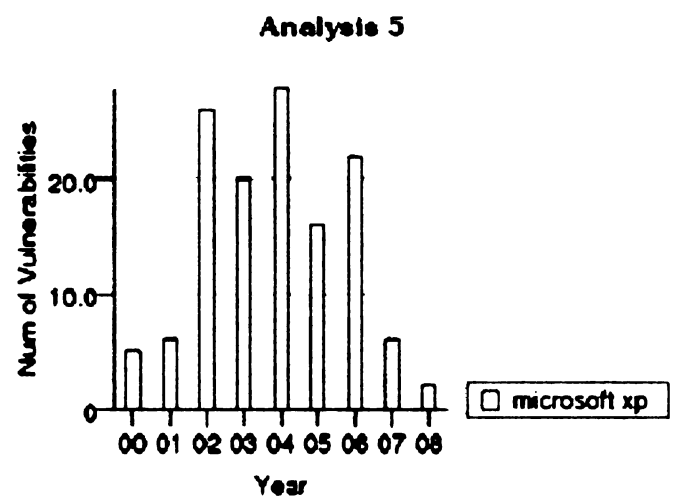
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	2	6	6	8	5	3	1	1	32
Percent	0.00	0.80	1.64	3.05	2.36	1.97	0.92	0.27	1.39	1.46

Analysis 4

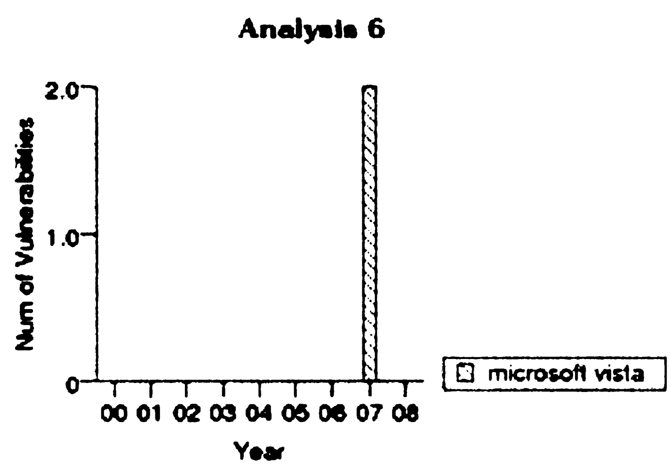


	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	0	0	0	0	0	0	2	0	2
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.55	0.00	0.09

Figure 5.2. VACT results searching for Microsoft windows xp and Microsoft windows vista within US-CERT

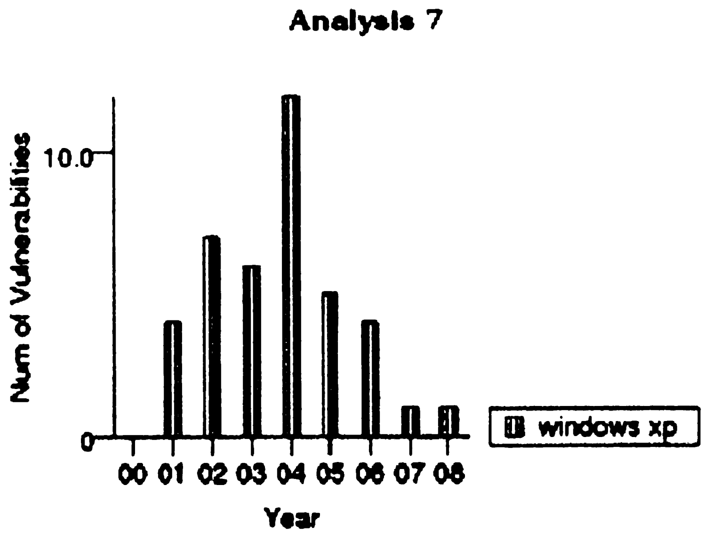


	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	5	6	26	20	28	16	22	6	2	131
Percent	16.13	2.41	7.12	10.15	8.26	6.30	6.75	1.64	2.78	5.96

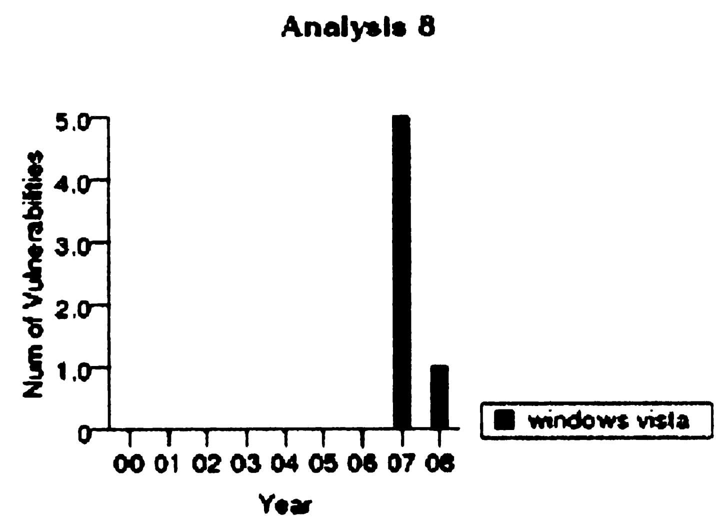


	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	0	0	0	0	0	0	2	0	2
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.55	0.00	0.09

Figure 5.3. VACT results searching for Microsoft xp and Microsoft vista within US-CERT



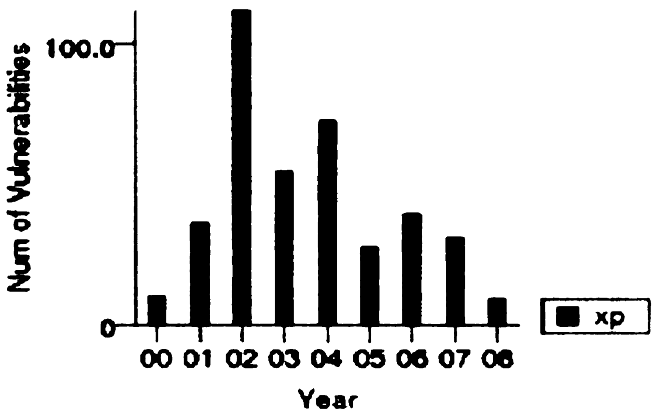
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	4	7	6	12	5	4	1	1	40
Percent	0.00	1.61	1.92	3.05	3.54	1.97	1.23	0.27	1.39	1.82



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	0	0	0	0	0	0	5	1	6
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.37	1.39	0.27

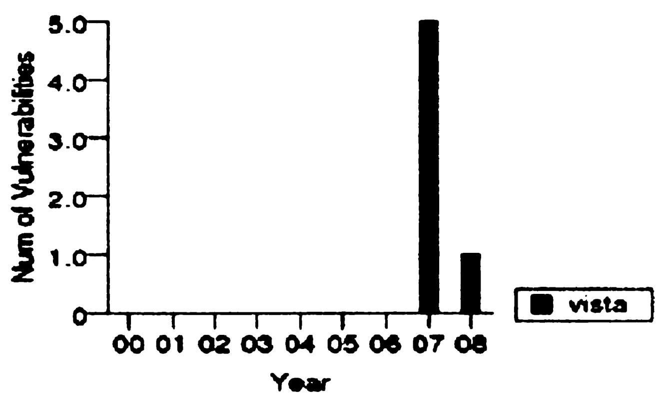
Figure 5.4. VACT results searching for windows xp and windows vista within US-CERT

Analysis 9



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	10	36	112	54	72	27	39	31	9	390
Percent	32.26	14.46	30.68	27.41	21.24	10.63	11.96	8.47	12.50	17.74

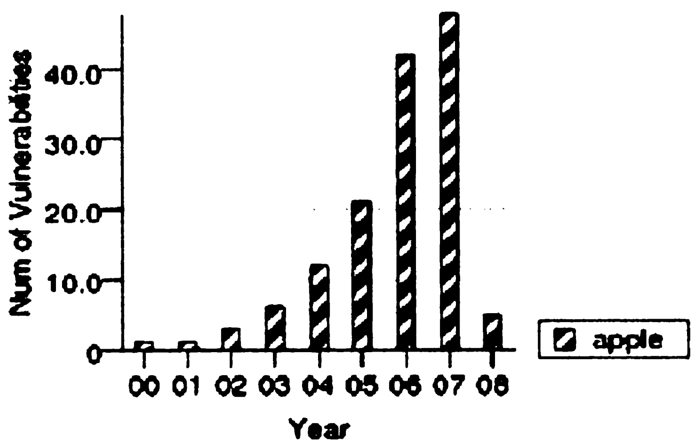
Analysis 10



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	0	0	0	0	0	0	5	1	6
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.37	1.39	0.27

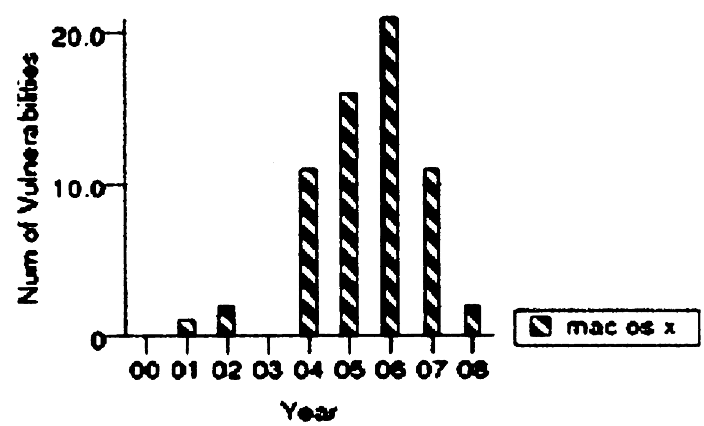
Figure 5.5. VACT results searching for xp and vista within US-CERT

Analysis 11



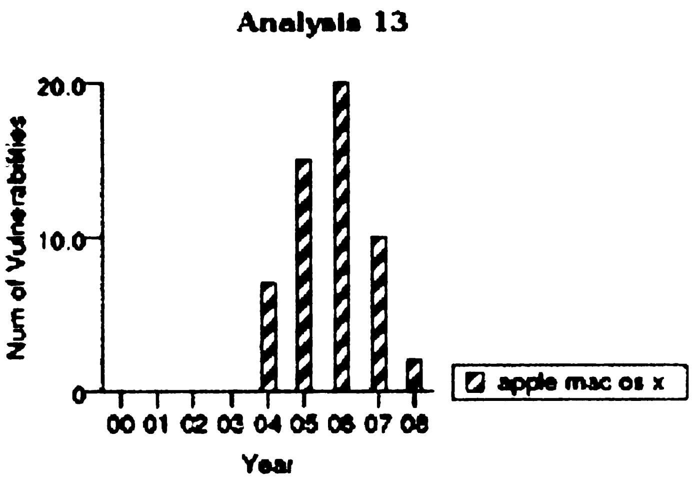
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	1	1	3	6	12	21	42	48	5	139
Percent	3.23	0.40	0.82	3.05	3.54	8.27	12.88	13.11	6.94	6.32

Analysis 12



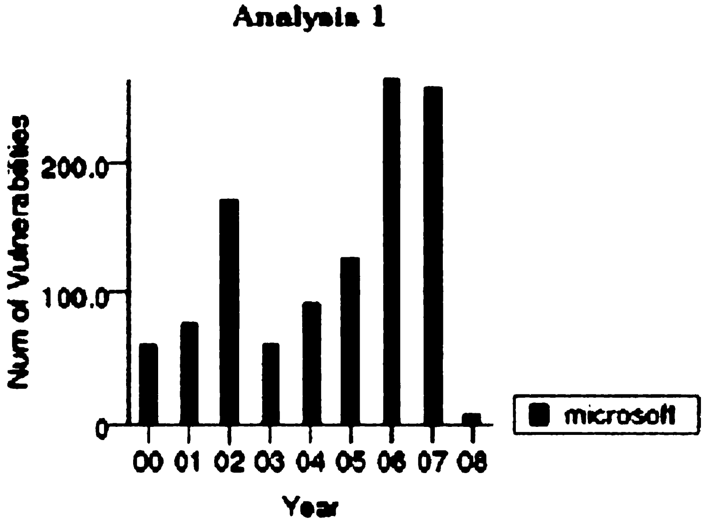
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	1	2	0	11	16	21	11	2	64
Percent	0.00	0.40	0.55	0.00	3.24	6.30	6.44	3.01	2.78	2.91

Figure 5.6. VACT results searching for apple and mac os x within US-CERT

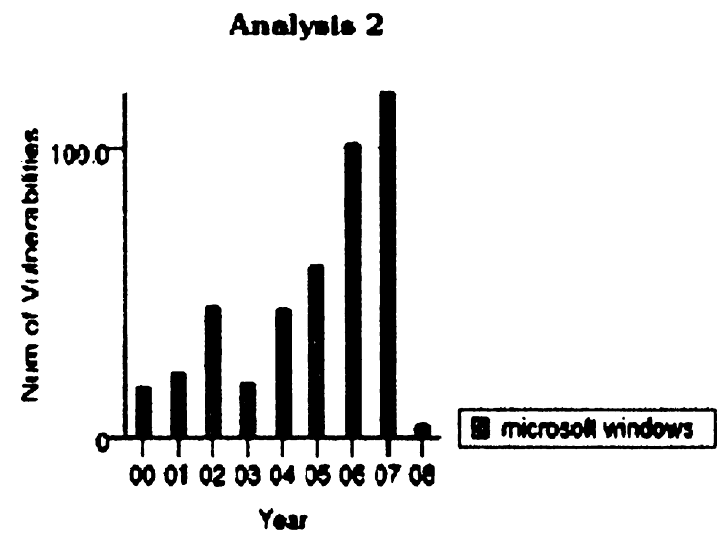


	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	31	249	365	197	339	254	326	366	72	2199
Matching	0	0	0	0	7	15	20	10	2	54
Percent	0.00	0.00	0.00	0.00	2.06	5.91	6.13	2.73	2.78	2.46

Figure 5.7. VACT results searching for apple mac os x within US-CERT



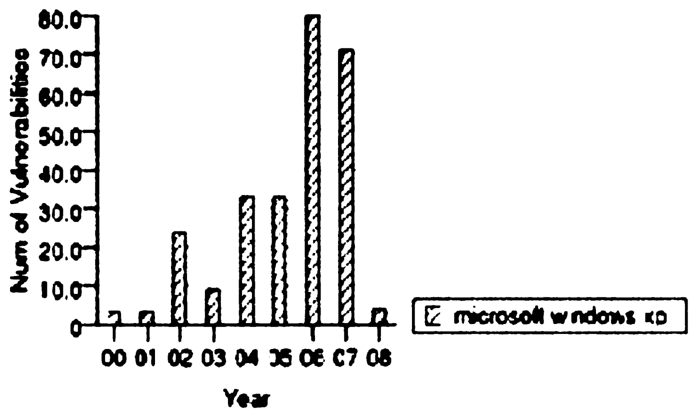
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	60	76	172	61	92	126	265	257	7	1116
Percent	5.88	4.53	7.93	3.96	3.71	2.52	3.98	3.90	3.57	4.08



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	17	22	45	19	44	59	102	120	4	432
Percent	1.67	1.31	2.07	1.23	1.78	1.18	1.53	1.82	2.04	1.58

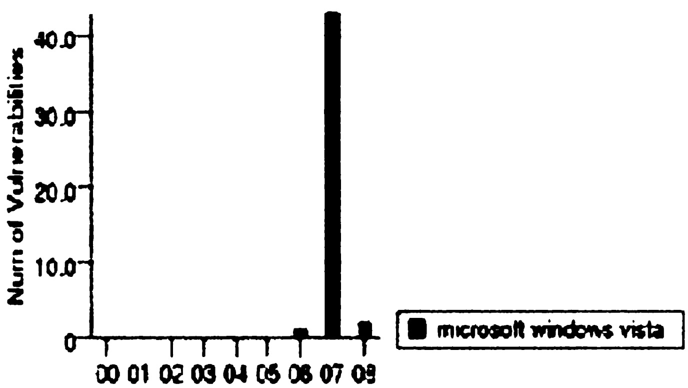
Figure 5.8. VACT results searching for Microsoft and Microsoft windows within NVD

Analysis 3



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	3	3	24	9	33	33	80	71	4	260
Percent	0.29	0.18	1.11	0.58	1.33	0.66	1.20	1.08	2.04	0.95

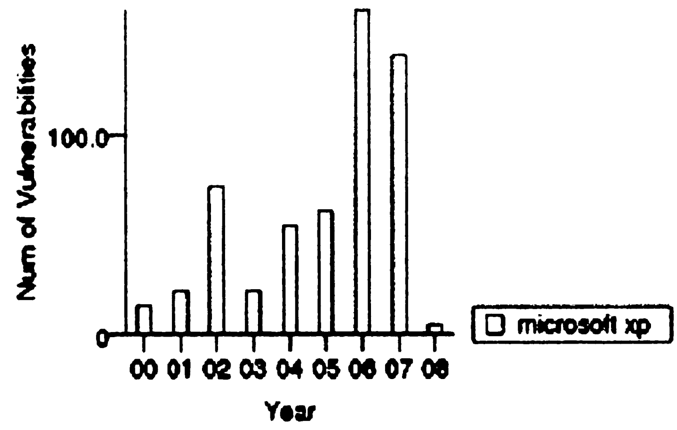
Analysis 4



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	0	0	0	0	0	0	1	43	2	46
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.65	1.02	0.17

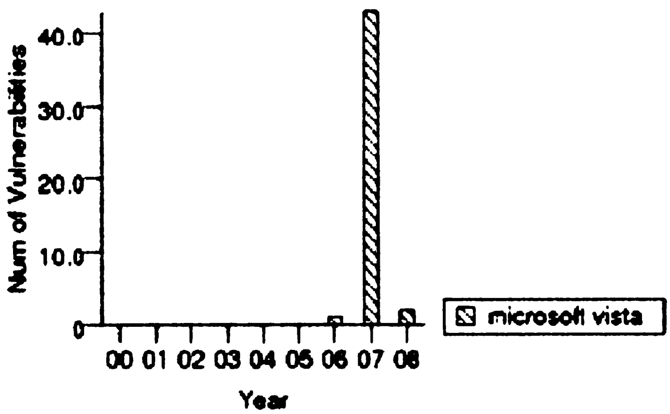
Figure 5.9. VACT results searching for Microsoft windows xp and Microsoft windows vista within NVD

Analysis 5



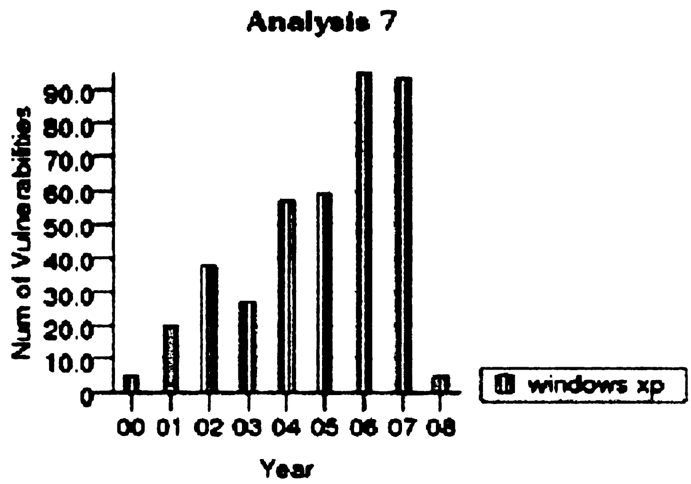
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	14	21	74	21	54	61	163	140	4	552
Percent	1.37	1.25	3.41	1.36	2.18	1.22	2.45	2.12	2.04	2.02

Analysis 6

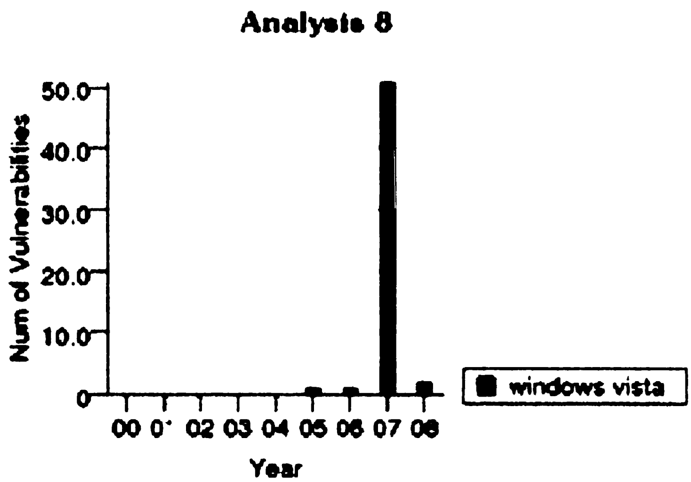


	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	0	0	0	0	0	0	1	43	2	46
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.65	1.02	0.17

Figure 5.10. VACT results searching for Microsoft xp and Microsoft vista within NVD



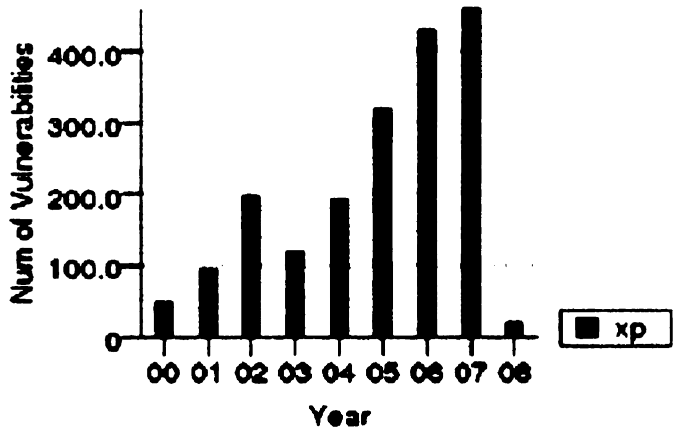
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	5	20	38	27	57	59	95	93	5	399
Percent	0.49	1.19	1.75	1.75	2.30	1.18	1.43	1.41	2.55	1.46



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	0	0	0	0	0	1	1	51	2	55
Percent	0.00	0.00	0.00	0.00	0.00	0.02	0.02	0.77	1.02	0.20

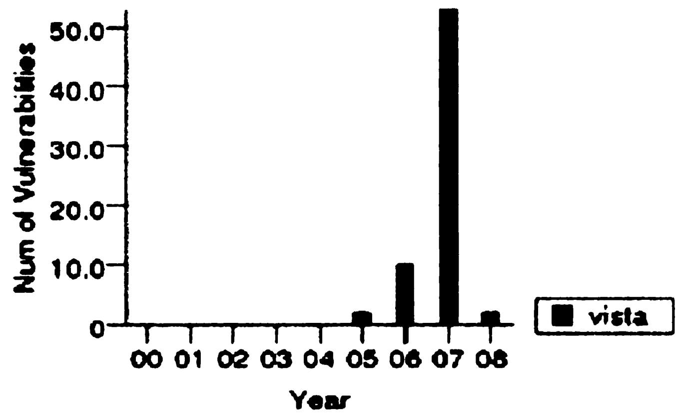
Figure 5.11. VACT results searching for windows xp and windows vista within NVD

Analysis 9



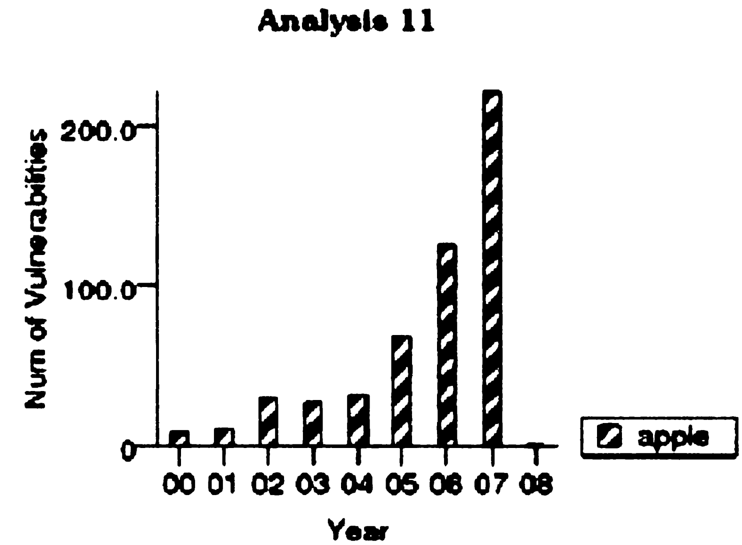
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	50	95	196	120	192	320	431	463	20	1887
Percent	4.90	5.66	9.03	7.79	7.75	6.39	6.48	7.02	10.20	6.90

Analysis 10

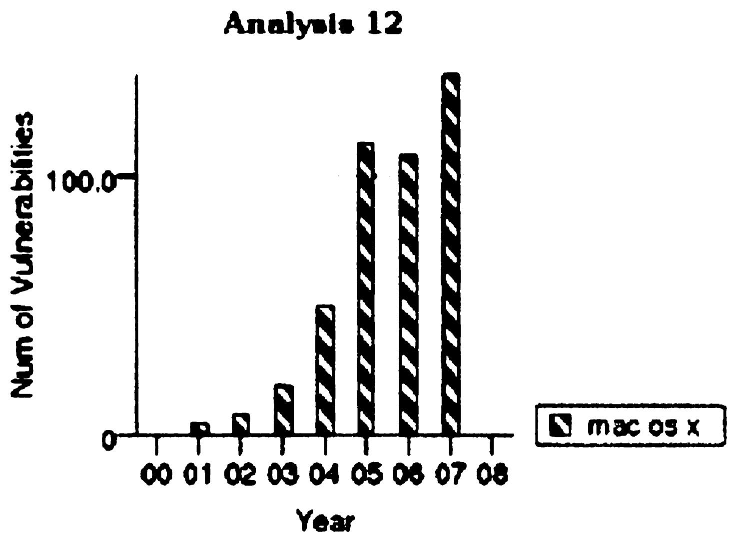


	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	0	0	0	0	0	2	10	53	2	67
Percent	0.00	0.00	0.00	0.00	0.00	0.04	0.15	0.80	1.02	0.25

Figure 5.12. VACT results searching for xp and vista within NVD

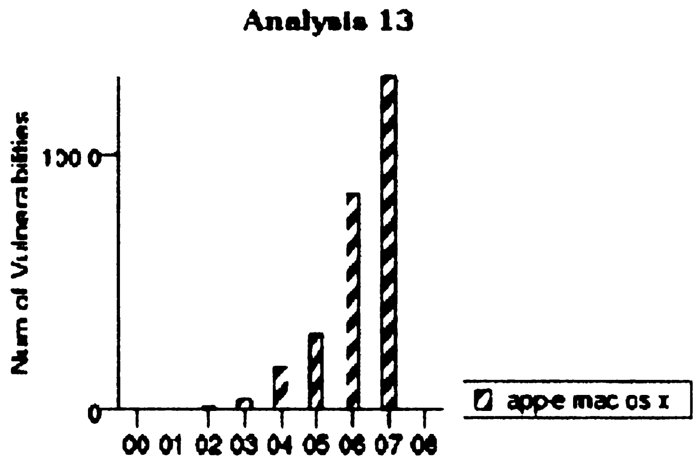


	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	8	10	30	27	32	68	126	222	1	524
Percent	0.78	0.60	1.38	1.75	1.29	1.36	1.89	3.37	0.51	1.92



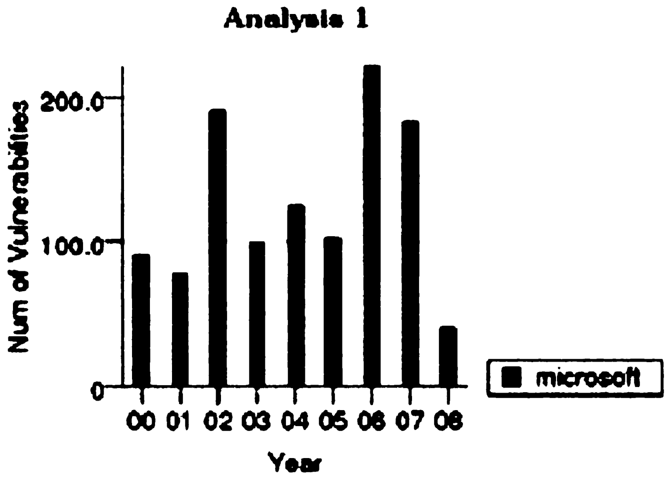
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	0	4	8	19	50	113	108	140	0	442
Percent	0.00	0.24	0.37	1.23	2.02	2.26	1.62	2.12	0.00	1.62

Figure 5.13. VACT results searching for apple and mac os x within NVD

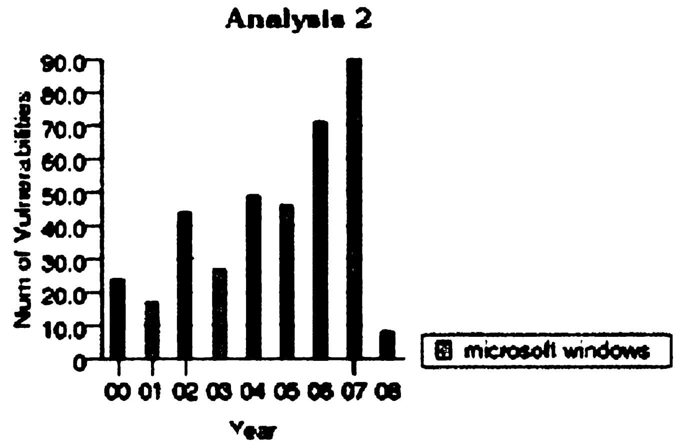


	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1020	1679	2170	1541	2478	5007	6655	6596	196	27342
Matching	0	0	1	3	16	29	84	131	0	264
Percent	0.00	0.00	0.05	0.19	0.65	0.58	1.26	1.99	0.00	0.97

Figure 5.14. VACT results searching for apple mac os x within NVD



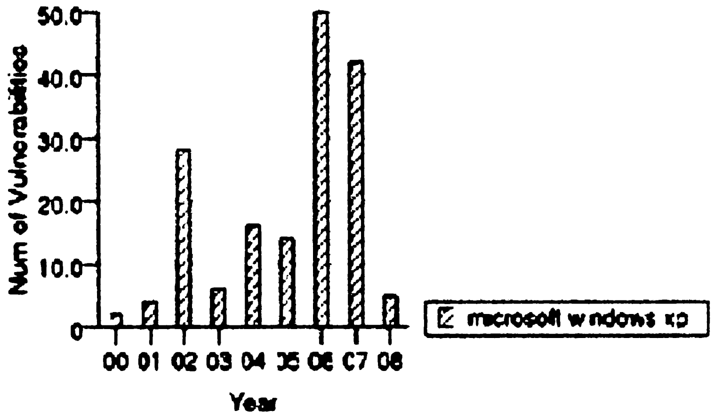
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	90	77	190	99	125	102	222	183	40	1128
Percent	6.62	4.60	8.19	3.61	2.62	1.37	2.10	2.19	1.70	2.71



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	24	17	44	27	49	46	71	90	8	376
Percent	1.76	1.01	1.90	0.99	1.03	0.62	0.67	1.08	0.34	0.90

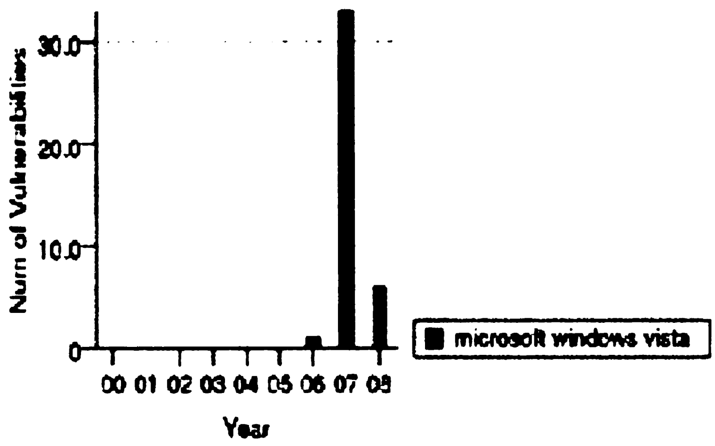
Figure 5.15. VACT results searching for Microsoft and Microsoft windows within OSVDB

Analysis 3



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	2	4	28	6	16	14	50	42	5	167
Percent	0.15	0.24	1.21	0.22	0.34	0.19	0.47	0.50	0.21	0.40

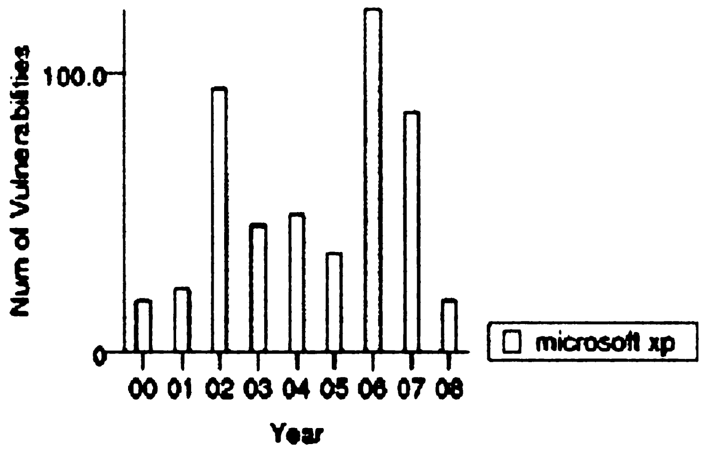
Analysis 4



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	0	0	0	0	0	0	1	33	6	40
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.40	0.25	0.10

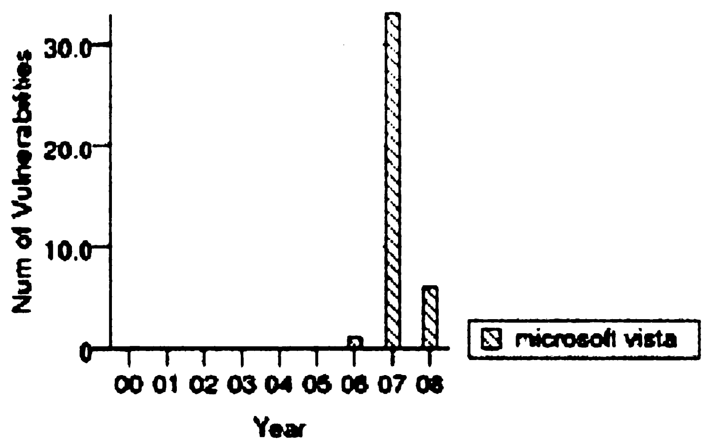
Figure 5.16. VACT results searching for Microsoft windows xp and Microsoft windows vista within OSVDB

Analysis 5



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	18	22	94	45	49	35	123	86	18	490
Percent	1.32	1.31	4.05	1.64	1.03	0.47	1.17	1.03	0.76	1.18

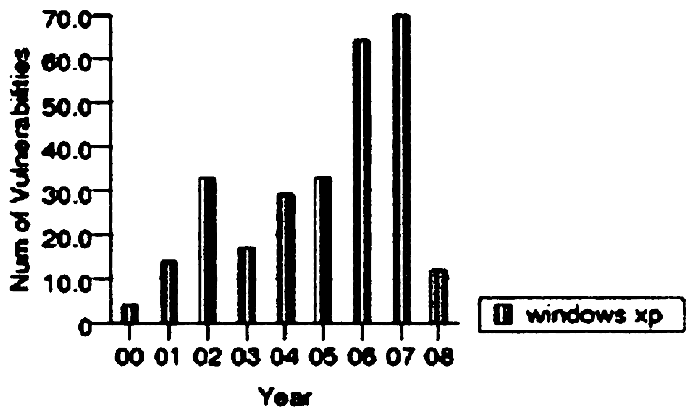
Analysis 6



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	0	0	0	0	0	0	1	33	6	40
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.40	0.25	0.10

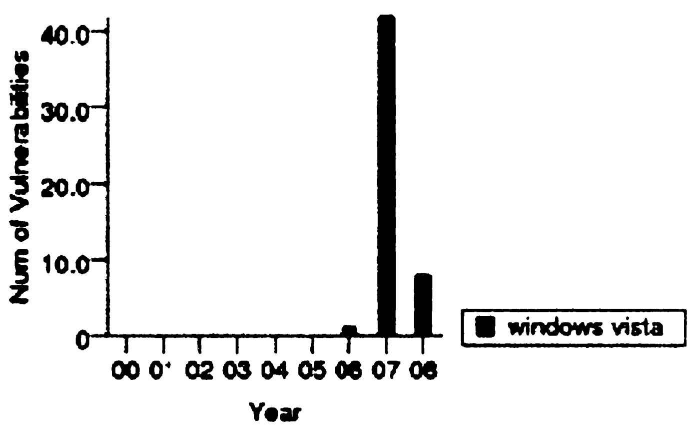
Figure 5.17. VACT results searching for Microsoft xp and Microsoft vista within OSVDB

Analysis 7



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	4	14	33	17	29	33	64	70	12	276
Percent	0.29	0.84	1.42	0.62	0.61	0.44	0.61	0.84	0.51	0.66

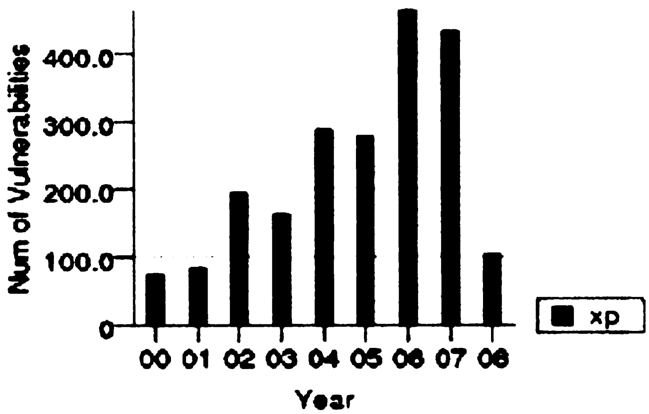
Analysis 8



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	0	0	0	0	0	0	1	42	8	51
Percent	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.50	0.34	0.12

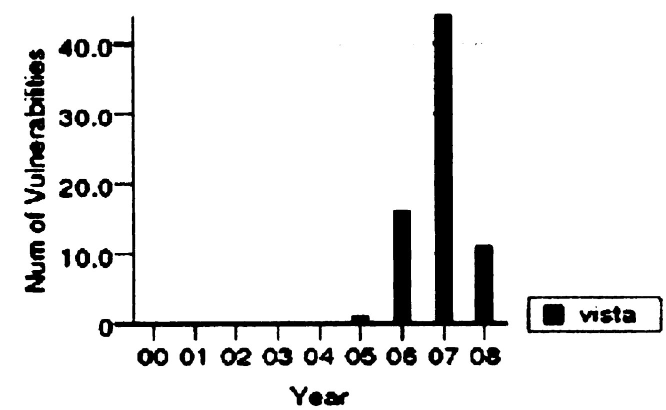
Figure 5.18. VACT results searching for windows xp and windows vista within OSVDB

Analysis 9



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	75	83	193	164	287	278	465	432	102	2079
Percent	5.51	4.96	8.32	5.99	6.01	3.72	4.41	5.18	4.33	5.00

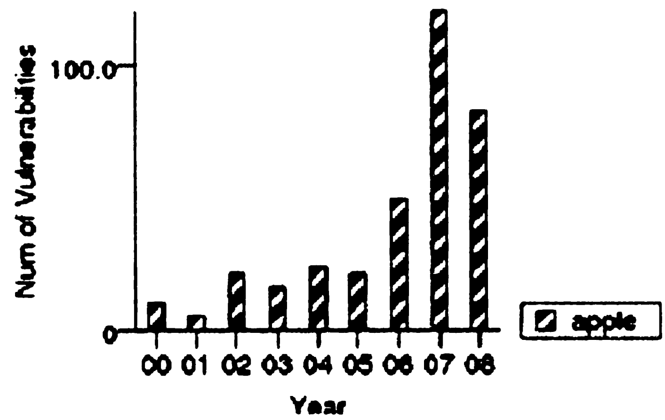
Analysis 10



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	0	0	0	0	0	1	16	44	11	72
Percent	0.00	0.00	0.00	0.00	0.00	0.01	0.15	0.53	0.47	0.17

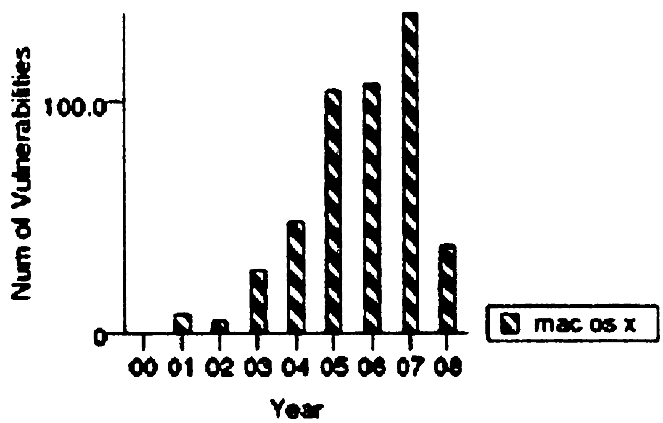
Figure 5.19. VACT results searching for xp and vista within OSVDB

Analysis 11



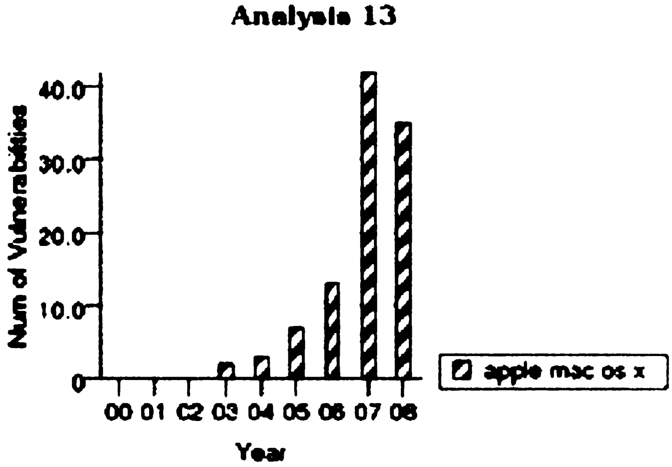
	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	10	5	21	16	24	21	50	121	83	351
Percent	0.74	0.30	0.91	0.58	0.50	0.28	0.47	1.45	3.52	0.84

Analysis 12



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	0	8	5	27	48	105	108	139	37	477
Percent	0.00	0.48	0.22	0.99	1.01	1.41	1.02	1.67	1.57	1.15

Figure 5.20. VACT results searching for apple and mac os x within OSVDB



	2000	2001	2002	2003	2004	2005	2006	2007	2008	total
Total	1360	1675	2320	2739	4773	7467	10547	8342	2357	41580
Matching	0	0	0	2	3	7	13	42	35	102
Percent	0.00	0.00	0.00	0.07	0.06	0.09	0.12	0.50	1.48	0.25

Figure 5.21. VACT results searching for apple mac os x within OSVDB

Chapter 6: Conclusion

Vulnerability Analysis and Classification Tool offers a unique way to find basic statistics on sets of vulnerabilities. There is currently no vulnerability database that is able to provide the statistical results on a user's vulnerability classification schema. By providing a customizable schema and basic framework, it can suit the needs of various users. The tool also saves the user disc space by downloading the needed vulnerabilities at each run. The tradeoff to downloading the necessary vulnerabilities comes as the download time is the constraint of the tools runtime.

APPENDICES

Setting up VACT

The steps listed below will help setup VACT.

1. Obtain a copy of Python. VACT was tested and run using Python 2.5.
2. Obtain a copy of Mod Python. Instructions for the setup and installation of Mod Python can be obtained at www.modpython.org.
3. Create a csv folder within the web directory. Give the folder APACHE_RUN_USER and APACHE_RUN_GROUP permissions. I found both to be www-data.
4. Download and install Pychart (Saito)
5. Copy files into web directory.

VACT Code

initial.py

```
######
#Call init to print out the initial user page for VACT
#Be sure to have the javascript.js file within the same directory
######
def init():
	text = """
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script language="javascript" src="javascript.js" type="text/javascript"></script>
<script language="javascript" src="contentloader.js" type="text/javascript"></script>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Vulnerability Search Tool</title>
</head>

<body onload="addSearch()">
<h3 align="center">Vulnerability Analysis and Classification Tool</h3>
<div style="width:330px; margin-left:auto; margin-right:auto;">
<form action="vact_ui.py/process" method="POST">
<div id="mainDiv" style="padding:10px; width:330px; text-align:left">
</div>
<input style="margin-left:20px" type="button" name="addSearch" value="Add Search"
onclick="mSearch()" />
<input type="hidden" value="-1" name="numSearch" id="numSearch" />
<input type="submit" value="Find Vulnerabilities" />
</form>
</div>
</body>
</html>
```

javascript.js

```
/************
mSearch is called initially when the webpage is first loaded.
The purpose is to call addSearch.
************/
function mSearch() {
	addSearch();
}

/*************
addSearch is called to add another Search setup to the webpage
*************/
function addSearch() {
 //Initail calls to establish where to add search and which search elements to add
 var div = document.getElementById('mainDiv');
 var ser = document.getElementById('numSearch');
 var num = (document.getElementById('numSearch').value -1) + 2;
 ser.value = num;

 //Create the new search
 var newdiv = document.createElement('div');
 var divIdName = 'searchDiv'+num;
 newdiv.setAttribute('id',divIdName);
 newdiv.innerHTML = makeDiv(num);
 div.appendChild(newdiv);
 minput = document.createElement('input');
 name = "wbcount"+num;
 minput.setAttribute('name',name);
 minput.setAttribute('id',name);
 minput.setAttribute('type','hidden');
 minput.setAttribute('value',0);
 div.appendChild(minput);

}

/************
addWord will add the users input into the search form
************/
function addWord(num){
	var id = document.getElementById('searchWords'+num);
	var text = document.getElementsByName('wb'+num)[0];
	var hidden = document.getElementById('wbcount'+num);
	if (text.value != ''){
		minput = document.createElement('input');
		name = 'wb' + num +"var"+hidden.value;
		minput.setAttribute('name',name);
		minput.setAttribute('type','hidden');
		minput.setAttribute('value',text.value);
		id.appendChild(minput);
		id.innerHTML += text.value + '<br />';
		text.value = '';
		hidden.value = Number(hidden.value) + 1;
	}
	return false;
```

```
}

/***************
makeDiv is called by addSearch. MakeDiv is responsible for creating the year and database content.
makeDiv takes a number as input to create the search for the associated number
***************/
function makeDiv(num){
	var text;
	text = '<input type="text" name="wb' + num + '" />  <input type="button" value="Add
Variable" onclick="return addWord(' + num + ')" />\n<br /><div id="searchWords' + num + '"></div><br
/>';
	var month = new
Array("January","February","March","April","May","June","July","August","September","October","Nove
mber","December");
	text += "Start Date: <select name='sdmonth" + num + "'>";
	for (m in month){
		n = parseInt(m) + 1
		text += '<option value="' + n + '">' + month[m] + '</option>\n';
	}
	text += "</select><select name='sdyear" + num + "'>";
	var startyear = 2008;
	var endyear = 1998;
	for (var i = startyear;i>=endyear;i--){
		text += '<option value="' + String(i) + '">' + String(i) + '</option>\n';
	}
	text += "</select><br />";
	text += "End Date:  <select name='edmonth" + num + "'>";
	for (m in month){
		n = parseInt(m) + 1
		text += '<option value="' + n + '">' + month[m] + '</option>\n';
	}
	text += "</select><select name='edyear" + num + "'>";
	var startyear = 2008;
	var endyear = 1998;
	for (var i = startyear;i>=endyear;i--){
		text += '<option value="' + String(i) + '">' + String(i) + '</option>\n';
	}
	text += "</select><br />";
	var sources = new Array("US Cert", "National Vulnerability Database", "Open Source
Vulnerability Database");
	for (s in sources){
		if (s == 0){
			text += '<input type="radio" checked="checked" name="source' + num + '"
value="' + sources[s] + '" />' + sources[s] + '<br />\n';
		}
		else{
			text += '<input type="radio" name="source' + num + '" value="' + sources[s] + '"
/>' + sources[s] + '<br />\n';
		}
	}
	text += "<br />";

	return text;
}
```

vact.py

```
import porterstem
import copy
import datetime
import time
import sys
from webinfo import *
from pychart import *

#####
#Function takes in list of db entries and years,
#Returns dictionary of vulnerabilities for each db
#####
def getDBvuls( db,year):
    #inital variable to help determine which db were selected by the user and the timeframe selected by the
user
    CERT = False
    NVD = False
    OSVDB = False
    nvdstart = 2008
    nvdend = 1999
    osstart = 2008
    osend = 1999
    certdict = { }
    nvddict = { }
    osdict = { }

    for i in range(0,len(db)):
        if db[i] == "US Cert":
            CERT = True
        elif db[i] == "National Vulnerability Database":
            NVD = True
            if int(year[i][0]) < nvdstart:
                nvdstart = int(year[i][0])
            if int(year[i][1]) > nvdend:
                nvdend = int(year[i][1])
        elif db[i] == "Open Source Vulnerability Database":
            OSVDB = True
            if int(year[i][0]) < osstart:
                osstart = int(year[i][0])
            if int(year[i][1]) > osend:
                osend = int(year[i][1])
        else:
            return "Improper variables specified %s" %(db[i])

    #Download the USCERT vulnerabilities
    if CERT:
        trial = []
        url = 'http://www.kb.cert.org/vuls/bypublished?open&start='
        base = 'http://www.kb.cert.org/vuls/id/'
        pat = [['VU#\d+', 3, 0, []],["\d+/\d+/\d+ \d+:\d+:\d+ \w+",0,0,[]] ]
        patt = [["Overview</H3></A>.+", 17, 0, ['<tt>','</tt>']]]
        num = 1
```

```
    x = True
    while(x):
      a = gethtml(url+str(num),pat,patt,base)
      x = a.bool
      if x:
        trial.append(a)
        a.start()
      else:
        del a
      num += 30

    for a in trial:
      a.join()

    #Populate dictionary with vulnerabilities
    #Key = vulnerability name, vulnerability date
    #Value = (set of vulnerability description,datetime element of date)
    for a in trial:
      for i in range(0,len(a.results[0])):
        try:
            certdict[a.results[0][i]+' , '+a.results[1][i]] =
(set(),datetime.date(int(a.results[1][i][6:10]),int(a.results[1][i][0:2]),int(a.results[1][i][3:5])))
            for word in a.results[2][i].split():
              certdict[a.results[0][i]+' , '+a.results[1][i]][0].add(word.lower())
        except:
            pass


       for a in trial:
         trial.remove(a)
         del a

  #Download the NVD vulnerabilities
  if NVD:

    trial = []
    url = "http://nvd.nist.gov/download/nvdcve-"
    entry = ['entry', ['name="[a-zA-Z0-9\-]+"',6,-1,[]], ['published="[0-9\-]+"', 11, -1, []] ]
    features = [ 'descript' ]
    #find which years need to be downloaded
    if not(nvdstart >= 2002 and nvdstart <= int(time.ctime()[-4:])):
      nvdstart = 2002
    if not(nvdend >= nvdstart and nvdend <= int(time.ctime()[-4:])):
      nvdend = int(time.ctime()[-4:])

    for num in range(nvdstart,nvdend+1):
      try:
            a = getxml(url+str(num)+".xml",entry,features)
        trial.append(a)
        a.start()
         except:
            pass

    for a in trial:
      a.join()
```

```
        #Populate dictionary with vulnerabilities
      #Key = vulnerability name, vulnerability date
      #Value = (set of vulnerability description,datetime element of date)

        for a in trial:
          for i in range(0,len(a.results[0])):
            try:
              nvddict[a.results[0][i][0]+' , '+a.results[1][i][0]] =
(set(),datetime.date(int(a.results[1][i][0][0:4]),int(a.results[1][i][0][5:7]),int(a.results[1][i][0][8:10])))
              for word in a.results[2][i].split():
                nvddict[a.results[0][i][0]+' , '+a.results[1][i][0]][0].add(word.lower())
            except (IndexError):
              pass


        for a in trial:
          trial.remove(a)
          del a

    #Download the OSVDB vulnerabilities
    if OSVDB:
      trial = []
      url = 'http://osvdb.org/browse/by_disclosure_date/'
      base = ''
      pat = [ ['style="">\d+',9,0,[]], ['Disclosed:[ 0-9\-]+',11,0,[]], ['Description:</span></p>[\n]<p>[()>\w\-
:</?\';=\".\*#,]+',0,0,[]] ]
      patt = []
      num = 1
      if osstart <= int(time.ctime()[-4:]):
        yr = osstart
      else:
        yr = int(time.ctime()[-4:])
      x = True
      if not (osend <= int(time.ctime()[-4:])):
        osend = int(time.ctime()[-4:])
      while( x ) :
        turl = url + str(yr)+ '?page=' + str(num)
        a = gethtml(turl,pat,patt,base)
        x = a.bool
        if x:
          trial.append(a)
          a.start()
        else:
          del a
        num += 1
        if not x and yr <= osend:
          num = 1
          yr += 1
          x = True

      for a in trial:
        a.join()

      #Populate dictionary with vulnerabilities
      #Key = vulnerability name, vulnerability date
      #Value = (set of vulnerability description,datetime element of date)
```

```
    for a in trial:
      for i in range(0,len(a.results[0])):
              try:
          osdict[a.results[0][i]+' , '+a.results[1][i]] =
(set(),datetime.date(int(a.results[1][i][0:4]),int(a.results[1][i][5:7]),int(a.results[1][i][8:10])))
          for word in a.results[2][i].split():
            osdict[a.results[0][i]+' , '+a.results[1][i]][0].add(word.lower())
        except (IndexError):
          pass

      for a in trial:
        trial.remove(a)
        del a

  return [certdict,nvddict,osdict]

##########
#searchVuls searches the downloaded vulnerabilities from getDBvuls with the users criteria
#The input takes the list of databases specified within the search, the list of vulnerabilities for getDBvuls,
#the list of months from the search within a tuple, the list of years from the search with a tuple, and the
#list of search words specified by the user.
#The output contains the list of the number of returned vulnerabilities from each search while totlist
contains the total number of vulnerabilities within each search
##########
def searchVuls(db,vuls,month,year,search):

  slist = []
  temp = []
  rlist = []
  totlist = []

  for i in range(0,len(db)):
    if int(month[i][0]) > 11 or int(month[i][0]) < 0:
      d1 = datetime.date(int(year[i][0])+1,1,1)
    else:
      d1 = datetime.date(int(year[i][0]),int(month[i][0]),1)
    if int(month[i][1]) > 10 or int(month[i][1]) < 0:
      d2 = datetime.date(int(year[i][1])+1,1,1)
    else:
      d2 = datetime.date(int(year[i][1]),int(month[i][1])+1,1)

    if db[i] == "US Cert":
      temp = strsearch(search[i],copy.deepcopy(vuls[0]),(d2,d1))
    elif db[i] == "National Vulnerability Database":
      temp = strsearch(search[i],copy.deepcopy(vuls[1]),(d2,d1))
    elif db[i] == "Open Source Vulnerability Database":
      temp = strsearch(search[i],copy.deepcopy(vuls[2]),(d2,d1))
    slist.append(temp)
    temp.start()

  for i in range(0,len(slist)):
    slist[i].join()

    if db[i] == "US Cert":
        y,x = compute(slist[i].content,i,vuls[0],year[i])
    elif db[i] == "National Vulnerability Database":
```

```
            y,x = compute(slist[i].content,i,vuls[1],year[i])
        elif db[i] == "Open Source Vulnerability Database":
            y,x = compute(slist[i].content,i,vuls[2],year[i])
        rlist.append(y)
        totlist.append(x)

    return rlist,totlist

##########
#makehtml constructs the output page for the user.
#The input includes the list of search values returned per year for each search (searchVuls rlist),
#the list of total values per year for each search(searchVuls totlist), and the list of years specified by the
user within the search
##########
def makehtml(slist,tlist,year):
    row0 = []
    row1 = []
    row2 = []
    row3 = []

    for x in range(0,len(slist)):
        tab0 = '<td ALIGN="left"> </td>'
        tab1 = '<td ALIGN="left">Total</td>'
        tab2 = '<td ALIGN="left">Matching</td>'
        tab3 = '<td ALIGN="left">Percent</td>'
        for y in range(0,(int(year[x][1])+1-int(year[x][0]))):
            tab0 += '<td ALIGN="right">%d</td>' %(int(year[x][0]) + y)
            tab1 += '<td ALIGN="right">%d</td>' %(tlist[x][y])
            tab2 += '<td ALIGN="right">%d</td>' %(slist[x][y])
            if not tlist[x][y] == 0:
                tab3 += '<td ALIGN="right">%.2f</td>' %(slist[x][y]/float(tlist[x][y])*100)
            else:
                tab3 += '<td>0</td>'
        tab0 += '<td ALIGN="right">total</td>'
        tab1 += '<td ALIGN="right">%d</td>' %(sum(tlist[x]))
        tab2 += '<td ALIGN="right">%d</td>' %(sum(slist[x]))
        try:
                tab3 += '<td ALIGN="right">%.2f</td>' %(sum(slist[x])/float(sum(tlist[x]))*100)
            except:
                tab3 += '<td ALIGN="right">0.00</td>'
        row0.append(tab0)
        row1.append(tab1)
        row2.append(tab2)
        row3.append(tab3)

    init = """
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>VACT Analysis</title>
</head>
<body>
    <h2 align="center">Results</h2>
"""
```

```
    top = """
<div>
    <h3 align="center" style="text-decoration:none"><a href="/csv/results%d.csv">Analysis %d</a></h3>
    <table>
        <tr>
            <td><img src="/csv/%d.png" width="490" height="312" border="0" alt=""></td>
            <td>
                <table>
"""

    table = """
                <tr>
                    %s
                </tr>
"""

    bottom = """
                </table>
            </td>
        </tr>
        </table>
</div>
"""

    end = """
<div align="center">
    <img src="/csv/all.png" width="980" height="624" border="0">
</div>
</body>
</html>
"""

    html = init
    for i in range(0,len(slist)):
        html += top % (i,i+1,i)
        html += table %(row0[i])
        html += table %(row1[i])
        html += table %(row2[i])
        html += table %(row3[i])
        html += bottom
    html += end

    return html

##########
#compute will turn the search results into CSV files associated by search and return the statistics per year
#The input for compute is the dictionary of search results, the search number, the dictionary of total
vulnerabilies, and the years for the search
#The output will be the CSV files with the csv webroot directory and the statistical files for the total
vulnerabilities and specified vulnerabilities
##########
def compute(mdict,snum,tdict,year):

    slist = []
    tlist = []
    diff = int(year[1]) - int(year[0])
```

```
  try:
    f = open(('/var/www/csv/results'+str(snum)+'.csv'),'w')
  except:
    f = open(str(snum)+'.csv','w')
  for i in range(int(year[0]),int(year[1])+1):
    slist.append(0)
    tlist.append(0)

  for key in mdict.keys():
    x = int(year[1]) - int(mdict[key][1].year)
    if x <= diff and x >=0:
      slist[x] += 1
    temp = ""
    for i in mdict[key][0]:
      temp = "%s %s" %(temp,i)
    temp = temp[1:]
    f.write( '%s,%s\n' %(key,temp.replace(',','')) )
  f.close()
  for key in tdict.keys():
    x = int(year[1]) - int(tdict[key][1].timetuple()[0])
    if x <= diff and x >=0:
      tlist[x] += 1
  slist.reverse()
  tlist.reverse()
  return slist,tlist

##########
#generateGraphs will compute the graphs of each search and then a graph containing all search results
#The input is the statistics that are returned from searchVuls for the search results and total results in addition to the search terms
#The output is a graph within the csv webroot directory named by the search number
##########
def generateGraphs(slist, year, search):
        theme.scale_factor = 2
        theme.reinitialize()
        data = []

        smin = int(min([year[i][0] for i in range(0,len(year))]))
        smax = int(max(([year[i][1] for i in range(0,len(year))])))
        alldata = []
        for i in range(smin,smax+1):
      alldata.append([str(i)[2:]])
        all_lab = []
        allmax = []

        for i in range(0,len(slist)):
      #make individual graphs
      ylist = []
      for j in range(int(year[i][0]),int(year[i][1])+1):
        ylist.append(str(j)[2:])
      data = zip(ylist,slist[i])
      try:
              mymax = int(max(slist[i]))
          except:
              mymax = 0
      allmax.append(mymax)
```

```
        if mymax == 0:
                mymax = 1
    lab = ""
    for x in search[i]:
      lab = "%s %s" %(lab,x)
    all_lab.append(lab)
    can = canvas.init("/var/www/csv/"+str(i)+".png")
    ar = area.T(x_coord = category_coord.T(data, 0), y_range = (0, mymax),
            x_axis = axis.X(label="Year"),
            y_axis = axis.Y(label="Num of Vulnerabilities"))

    ar.add_plot(bar_plot.T(data = data, label = lab))
    ar.draw(can)
    can.close()

        #compute data for combined graph

    for i in range(0,len(slist)):
      j = 0
      for k in range(0,int(year[i][0])-smin):
        alldata[k].append(0)
      for k in range(int(year[i][0])-smin,int(year[i][1])-smin+1):
        alldata[k].append(slist[i][j])
        j+=1
      for k in range(int(year[i][1])-smin+1,smax-smin+1):
        alldata[k].append(0)

  theme.scale_factor = 4
  theme.reinitialize()
  theme.get_options()

  can = canvas.init("/var/www/csv/all.png")
      if not (max(allmax) > 0):
              allmax.append(1)
  chart_object.set_defaults(area.T, size = (300,240), y_range = (0,max(allmax)),
                    x_coord = category_coord.T(alldata, 0))
  chart_object.set_defaults(bar_plot.T, data = alldata)

  ar = area.T(x_axis=axis.X(label="Year"),
        y_axis=axis.Y(label="Num of Vulnerabilities"))

  for i in range(0,len(all_lab)):
    ar.add_plot(bar_plot.T(label=all_lab[i], hcol=i+1, cluster=(i,len(all_lab))))

  ar.draw(can)
      can.close()


##########
#User can input the search results in text form
#The results must list the databases, years, months, and search terms
#All input must be separated with <>
#If the search terms contain multiple words per line separate the words with $%$
##########
if len(sys.argv) == 5:
        #split up the input
```

```
db = sys.argv[1].split('<>')
years = []
for i in sys.argv[2].split('<>'):
        years.append( (i.split(',')[0],i.split(',')[1])     )
months = []
for i in sys.argv[3].split('<>'):
        months.append( (i.split(',')[0],i.split(',')[1]) )
strings = []
for i in sys.argv[4].split('<>'):
        strings.append( i.split('$%$') )

#Call the series of functions generate the search
vuls = getDBvuls( db,years)

rlist,tlist = searchVuls(db,vuls,months,years,strings)

generateGraphs(rlist, years, strings)

out = makehtml(rlist,tlist,years)
```

vact_ui.py

```
import initial
import os
import time
import string

#The initial search page 127.1.1.1/vact_ui
def index(req):
	return initial.init();

#The form within the initial search page is set to send results to /vact_ui/process.py
#process will sort through the user for and send it to the search then return the results
def process(req):

	searches = int(req.form['numSearch'])
	strings = []
	sources = []
	years = []
	months = []
	for i in range(0,searches+1):
		sources.append(req.form['source'+str(i)])
		years.append( (req.form['sdyear'+str(i)] + ',' + req.form['edyear'+str(i)]) )
		months.append( (req.form['sdmonth'+str(i)] + ',' + req.form['edmonth'+str(i)]) )
		temp = []
		try:
			for j in range(0,int(req.form['wbcount'+str(i)])):
				#req.form['wb'+str(i)+'var'+str(j)]
				try:
					temp.append(str(req.form['wb'+str(i)+'var'+str(j)]).lower())
				except:
					return "You are missing a search word in search %d" %(i+1)
		except:
			return "You are missing a search word in search %d" %(i+0)
		strings.append(string.join(temp,'$%$'))

	req.content_type = 'text/html'

	arg1 = string.join(sources,'<>')
	arg2 = string.join(years,'<>')
	arg3 = string.join(months,'<>')
	arg4 = string.join(strings,'<>')
	#Uncomment to print search start time
	#req.write(time.ctime())
	req.write("python /var/www/vact.py '%s' '%s' '%s' '%s'" %(arg1,arg2,arg3,arg4))
	out =os.popen("python /var/www/vact.py '%s' '%s' '%s' '%s'" %(arg1,arg2,arg3,arg4))
	temp = out.read()
	temp = str(len(out.readlines())) + temp
	out.close()
	return temp
```

webinfo.py

```
import re
import urllib
import xml.dom.minidom
import datetime
from threading import Thread

##########
#gethtml is made to crawl through a website
#url is the website that it is initially crawling
#f_patterns are the re patterns that should be parsed from the initial url
#s_patterns are the re patterns tha should be parsed from the secondary search
#baseurl is the url to which results from the first page can be passed to download another page
#gethtml works by first downloading the content of a page and searching for re values specified by the user
#In our case it is looking for the vulnerability date, ID number and description
#Because not all vulnerability descriptions are listed on the initial page, the function will
#download and strip the description of a page specified from the first
##########
class gethtml(Thread):
  def __init__ (self,url,f_patterns,s_patterns,baseurl):
    Thread.__init__(self)
    self.url = url
    self.f_patterns = f_patterns
    self.s_patterns = s_patterns
    self.baseurl = baseurl
    self.bool = True
    try:
          data = getsource(self.url)
       except:
          try:
                data = getsource(self.url)
          except:
             data = ""
    results = []
    for i in range(0,len(self.f_patterns)):
      #print self.patterns[i]
      results.append(stripPattern(data,self.f_patterns[i]))
      #print 'ok\n'
    if len(results[0]) == 0:
      self.results = []
      self.bool = False
    else:
      self.results = results

  def run (self):
    if not self.bool:
      return
    temp = []
    for j in range(0,len(self.s_patterns)):
      for i in range(0,len(self.results[0])):
      #print self.baseurl+results[0][i]
        try:
          temp.append(stripPattern(getsource(self.baseurl+self.results[0][i]),self.s_patterns[j])[0])
        except (IndexError):
          temp.append("")
```

```
        #print 'done\n'
        self.results.append(temp)
        temp = []

#####
#getxml is designed to download and parse xml from a website
#The main goal is to find the vulnerability information set by the user
#The url is where to download the file
#The entry is where to find the various xml entries. Each vulnerability is contained within an entry
#features contains the real expression values to parse the features from each entry
#####
class getxml(Thread):
  def __init__ (self,url,entry,features):
    Thread.__init__(self)
    self.url = url
    self.entry = entry
    self.features = features
        self.results = []

  def run (self):
    data = getsource(self.url)
    results = []
    try:
      temp = xml.dom.minidom.parseString(data)
    except:
      self.results = [[]]
      return

    temparray = []
    entrys = temp.getElementsByTagName(self.entry[0])
    for i in range(1,len(self.entry)):
      for e in entrys:
        temparray.append(stripPattern(e.toxml('utf-8'),self.entry[i]))
      results.append(temparray)
      temparray = []

    for i in range(0,len(self.features)):
      for e in entrys:
        x = e.getElementsByTagName(self.features[i])
        if x == []:
          temparray.append('unknown')
        else:
          cattext = ''
          for child in x[0].childNodes:
            cattext += child.toxml('utf-8')
          temparray.append(cattext)
      results.append(temparray)
      temparray = []
    self.results = results

#########
#strsearch takes in the search specified by the user, the dictionary to search within and the dates specified by the user
#search is a list of the search words, content is a dictionary with the key name , date of the vulnerability values is tuple(set(description),datetime.date object)
#########
```

```
class strsearch(Thread):
   def __init__ (self,string,content,date=(datetime.date(2010,12,31),datetime.date(1970,1,1))):
      Thread.__init__(self)
      self.string = string
      self.content = content
      self.date = date
   def run(self):
         for rows in self.string:
            logic = makeSets(rows)
            print logic
        for desc in self.content.keys():
          try:
                    if not (eval(logic)):
                           self.content.pop(desc)
             elif self.content[desc][1] > self.date[0] or self.content[desc][1] < self.date[1]:
                self.content.pop(desc)
          except:
             #print desc
             #print self.content[desc][0]
                    self.content.pop(desc)

###########
#Not currently implemented within the code
#modifystring will modify the string to a users requirements
#the input is a list of strings with a list of replacements
#For each word within the list it will make any replacements
###########
def modifystring(lstring,replacements):
   temp = []
   p = PorterStemmer()
   for string in lstring:
      sentence = ''
      for word in string.split():
         word = word.lower()
         for key in replacements.keys():
            word = word.replace(key,replacement[key])
         if (False): # change to true to get porterstem of word
            word = p.stem(word,0,len(word)-1)
         sentence = sentence + word + ' '
      temp.apppend(sentence)
   return temp

####################################
#Retrives a webpage or file from the web
#Input the URL of the webpage
#Return the page content
####################################
def getsource(url):
   file = urllib.urlopen(url)
   data = file.read()
   file.close()
   return data

##################################################
#Finds a repeating pattern within the text
#Input string of text, real expression pattern, beginning result concatination,
```

```
#ending result concatination, symbols which need removed
#Return List of requested pattern found within text
#####################################################

def stripPattern(string, ex_list):
    Pattern = re.findall(ex_list[0],string)
    for r in range(0,len(Pattern)):
        if (ex_list[2] == 0):
            Pattern[r] = Pattern[r][ex_list[1]:]
        elif not (ex_list[1] == 0 and ex_list[2] == 0):
            Pattern[r] = Pattern[r][ex_list[1]: ex_list[2]]
        for i in ex_list[3]:
            Pattern[r] = Pattern[r].replace(i,"")
    return Pattern

####################
#Will make an array of sets and the logic for search from user input
#makeSets is called by strsearch
#It takes in the users input and converts it into a logical comparison to make within the search dictionary
####################

def makeSets(string):
    wordlist = re.findall('[()>\w\-:</?\';=\".\*#,]+',string)
    condcount = 0
    wordcount = 0
    logic = ""
    for word in wordlist:
            #print word
        while word[0] == '(':
                logic = logic + '('
                word = word[1:]
            while word[0] == ')':
                logic = logic + ')'
                word = word[1:]
            if len(word) == 0:
                pass
            elif word.lower() == 'and':
                logic = logic + ' and '
                condcount += 1
            elif word.lower() == 'or':
                logic = logic + ' or '
                condcount += 1
            elif (condcount < wordcount):
                count = 0
                while word[len(word)-1] == ')':
                        word = word[:-1]
                        count += 1
                if not(len(word) == 0):
                    logic = logic + ' and "%s" in self.content[desc][0]' %(word)
                for i in range(0,count):
                        logic = logic + ')'
            else:
                count = 0
                while word[len(word)-1] == ')':
                        word = word[:-1]
                        count += 1
```

```
        logic = logic + '"%s" in self.content[desc][0]' %(word)
        for i in range(0,count):
            logic = logic + ')'
        wordcount += 1

    return logic
```

BIBLIOGRAPHY

"About CCE." Common Configuration Enumeration. 18 Mar 2008. The MITRE Corporation. 15 Mar 2008 <http://cce.mitre.org/about/index.html>.

Bishop, Matt, "Vulnerabilities Analysis," Proceedings of the Symposium on Recent Advances in Intrusion Detection (Sep. 1999): 125–36

Christey, Steve, Robert A. Martin. " Vulnerability Type Distributions in CVE." Common Weakness Enumeration. The MITRE Corporation. 15 Mar 2008 <http://cwe.mitre.org/documents/vuln-trends/index.html>.

"Common Vulnerabilities and Exposures." Common Vulnerabilities and Exposures. 14 Apr 2008. The MITRE Corporation. 15 Mar 2008 <http://cve.mitre.org/>.

"Common Weakness Enumeration." Common Weakness Enumeration. 6 May 2008. The MITRE Corporation. 15 Mar 2008 <http://cwe.mitre.org/>.

Engle Sophie, Sean Whalen, Damien Howard, and Matt Bishop, "Tree Approach to Vulnerability Classification", Technical Report CSE-2006-10, Dept. of Computer Science, University of California at Davis, Davis, CA 95616-8562 (May 2006).

"How We Build the CVE List." Common Vulnerabilities and Exposures. 14 April 2008. The MITRE Corporation. 15 Mar 2008 <http://cve.mitre.org/cve/identifiers/build.html>.

"National Vulnerability Database." National Vulnerability Database. NIST. 15 Mar 2008 <http://nvd.nist.gov/nvd.cfm?advancedsearch>.

"NVD Common Vulnerability Scoring System Support v2." National Vulnerability Database. 28 July 2007. NIST. 15 Mar 2008 <http://nvd.nist.gov/cvss.cfm>.

"OSVDB GSoC 2008 Project Ideas." OSVDB Blog. 3 Mar 2008. OSVDB. 20 Mar 2008 <http://osvdb.org/blog/?p=231>.

"OSVDB: The Open Source Vulnerability Database." OSVDB: The Open Source Vulnerability Database. OSVDB. 15 Mar 2008 <http://osvdb.org/search/advsearch>.

"Process." Common Weakness Enumeration. 11 Sep 2007. The MITRE Corporation. 15 Mar 2008

Saito, Yasushi. "Pychart." Pychart. 15 Mar 2008 <http://home.gna.org/pychart/>.

"Search Advisory, Vulnerability, and Virus Database." Search Advisory, Vulnerability, and Virus Database. Secunia. 15 Mar 2008 <http://secunia.com/search/?search=&adv_search=1>.

"US-CERT Vulnerability Notes Database." US-CERT Vulnerability Notes Database. 2 July 2007. US-CERT. 15 Mar 2008 <http://www.kb.cert.org/vuls>.