

ABSTRACT

Title of Thesis: **DEVELOPMENT OF A HOST VULNERABILITY
CHECKING TOOL AND DATA ANALYSIS OF HOST
VULNERABILITIES**

Anil Sharma, Master of Science, 2004

Thesis Directed by: Dr. Michel Cukier
Center for Risk and Reliability,
Department of Mechanical Engineering

Vulnerabilities can be classified based on their location as application vulnerabilities, network vulnerabilities and host vulnerabilities. This thesis focuses on host vulnerabilities and can be divided in two parts. The first part describes Ferret, a new tool developed for checking host vulnerabilities. Ferret helps system administrators by quickly finding vulnerabilities that are present on a host. It is designed and implemented in a modular way: a different plug-in module is used for a specific vulnerability checked, and each possible output format is specified by a plug-in module. Ferret is freely available open-source software implemented in Perl. The second part of the thesis talks about the data analysis of host vulnerabilities. This exercise was conducted on two sets of data collected extensively for the purpose to understand how host vulnerabilities are evolving over a period of time.

**DEVELOPMENT OF A HOST VULNERABILITY CHECKING TOOL AND
DATA ANALYSIS OF HOST VULNERABILITIES**

by

Anil Sharma

**Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2004**

Advisory Committee:

Professor Michel Cukier, Chair

Professor Carol Smidts

Professor Marvin V Zelkowitz

ACKNOWLEDGEMENTS

It has been a wonderful experience for the past two years working with my advisor Dr Michel Cukier. He has been a constant source of encouragement and inspiration during various tough times and helped me to maintain the cool temperament during the research.

I would like to thank the co-developers of Ferret, Jason R. Martin and Nitin Anand for fruitful meetings and brainstorming sessions. I would like to thank Dr William H. Sanders for permitting a close collaboration with the University of Illinois, Urbana-Champaign. I would like to especially thank Tod Courtney for his help in testing and providing feedback on Ferret. I also thank Michael Chan, Loren Heal and Gabriel Lopez-Walle for allowing us to collect data on their networks at the University of Illinois. Special thank goes to Dr Yves Deswarte at LAAS-CNRS and Rodolphe Ortalo for providing the data on host vulnerabilities.

I am grateful to DARPA and NSF for funding the research project. The research described in this thesis was funded by DARPA contract F30602-00-C-0172 and NSF CAREER award 0237493.

A word of thanks also goes to my laboratory mate Melody Djam and Susmit Panjwani. They have been very cooperative and enlivened the work environment in the laboratory.

TABLE OF CONTENTS

LIST OF TABLES.....	v
LIST OF FIGURES.....	vi
Chapter 1: Introduction.....	1
Chapter 2: Ferret: A Host Vulnerability Checking Tool.....	7
2.1 Introduction.....	7
2.2 Ferret Design Goals.....	7
2.3 Ferret Detailed Design.....	9
2.3.1 Architecture Overview.....	10
2.3.2 Ferret Core.....	11
2.4 Vulnerability Checking Plug-ins.....	12
2.4.1 Options for Running the Vulnerability Checking Plug-ins.....	12
2.4.2 Groups of Vulnerability Checking Plug-ins.....	12
2.5 Output Plug-ins.....	16
2.5.1 Output Plug-in with the First Mode.....	16
2.5.2 Output Plug-in with the Second Mode.....	16
2.6 Comments on the Ferret Code.....	17
2.7 Conclusions.....	19
Chapter 3: Host Vulnerabilities Data Analysis	20
3.1 Introduction.....	20
3.2 Definition of Measures.....	21
3.3 Data Analysis: LAAS-CNRS.....	25
3.3.1 Introduction.....	25
3.3.2 Working Methodology.....	30

3.3.3 Results and Observations.....	31
3.3.4 Conclusions.....	41
3.4 Data Analysis: UIUC.....	42
3.4.1 Introduction.....	42
3.4.2 Working Methodology.....	42
3.4.3 Results and Observations.....	44
3.4.4 Conclusions.....	47
3.5 Comparison of the Two Data Analysis.....	48
Chapter 4: Conclusions and Future Work.....	50
Appendixes.....	51
Appendix A.....	52
Appendix A.1 Source code of Ferret Core.....	52
Appendix A.2 Source code of Vulnerability Checking Plug-ins.....	66
Appendix A.3 Source code of Output Plug-in.....	72
Appendix B.....	74
Appendix B.1 Plots obtained for Group A.....	74
Appendix B.2 Plots obtained for Group B.....	75
Appendix B.3 Plots obtained for Group E.....	76
Appendix B.4 Plots obtained for Group G.....	77
Appendix B.5 Plots obtained for Group H.....	78
Appendix B.6 Users Related Charts.....	79
Appendix C.....	80
Appendix C.1 Perl Script to Transfer Ferret Output Data to Mysql Database.....	80
Appendix C.2 Perl Script to Query Mysql Database.....	83
Appendix C.3 Macro to Plot Graphs in MS-Excel.....	85

References.....87

LIST OF TABLES

Table 3.1: Types of Vulnerabilities and their Groups.....	28
Table 3.2: Estimation for the Measure $AI_{s,v}$ and $L_{s,v}$	32
Table 3.3: Fields in the Mysql Table.....	43
Table 3.4: Hosts and Number of Vulnerabilities Found at UIUC.....	46
Table 3.5: Breakdown of Vulnerabilities Found on Various Hosts at UIUC.....	47

LIST OF FIGURES

Figure 2.1: Architecture Overview of Ferret.....	10
Figure 3.1: Time Intervals between Subsequent Readings.....	25
Figure 3.2: Total Number of Users as a Function of Time.....	26
Figure 3.3a: New Users added to the System.....	26
Figure 3.3b: New Users added to the System.....	26
Figure 3.4: Users removed from the System.....	27
Figure 3.5: $T_v(t)$ for Group A.....	33
Figure 3.6: $T_v(t)/ U_p(t)$ for Group A.....	33
Figure 3.7: $T_v(t)$ for Group B.....	35
Figure 3.8: $T_v(t)/ U_p(t)$ for Group B.....	35
Figure 3.9: $N_{cv}(t)$ for Group B.....	36
Figure 3.10: $R_{cv}(t)$ for Group B.....	36
Figure 3.11: $T_v(t)$ for Group D.....	37
Figure 3.12: $T_v(t)/ U_p(t)$ for Group D.....	37
Figure 3.13: $T_v(t)$ for Group F.....	38
Figure 3.14: $T_v(t)/ U_p(t)$ for Group F.....	39
Figure 3.15: $T_v(t)$ for Group G.....	40
Figure 3.16: $T_v(t)/ U_p(t)$ for Group G.....	40
Figure 3.17: $N_{cv}(t)$ for Group G.....	40
Figure 3.18: $R_{cv}(t)$ for Group G.....	41
Figure 3.19: $T_v(t)$ for Fifth Group.....	45

Chapter 1: Introduction

Operating systems combine building blocks like identification and authentication, access control and auditing to provide a coherent set of security controls. There is a general pattern to the way security controls are organized in existing operating systems. For example, both the information about users and the privileges granted to a user to access system resources can be stored in users account. Identification and authentication may verify a user's identity, allowing a system to associate the user's privileges with any process started by the user. Permissions on the resources can be set by the system manager or the owner of the resource. When deciding whether to grant or deny an access request, the operating system may refer to the user's identity, the user's privileges, and the permissions of the object. Similarly audit logs may be set up to detect and record the unauthorized actions of the users. But all these security features are not of much use till they are used properly. So the configuration of the operating system is an important issue.

This thesis focuses on UNIX systems and specifically targets host-based vulnerabilities. Here the term "UNIX" symbolizes a group of operating systems from different manufacturers and does not represent a single operating system such as Linux, FreeBSD, NetBSD, and OpenBSD [18]. These operating systems tend to run on Intel-based hardware. And there are also commercial UNIX operating systems such as Sun Microsystems Solaris, Hewlett-Packard HP-UX, IBM AIX, and Silicon Graphic IRIX [18]. Each of these commercial operating systems will only run on their own company's hardware. Many features are common to all of these operating systems. Examples are the use of a bourne shell (the command programming language used to provide an interface to the UNIX operating system), the ps command (the process status command to know the id of a process), or the X Window System (the graphics system used on UNIX systems). There are of course differences between these operating systems. For example, comparing a list of the currently running daemons (the processes that run in the background and are not associated with any terminal) between any of the operating systems above will reveal many differences in the names and number of daemons [17]. Likewise when comparing vulnerabilities between UNIX systems, some vulnerabilities

are common to almost all UNIX systems and other vulnerabilities are unique to a particular UNIX system [17].

UNIX is a multi-user, multi-tasking operating system. Multi-user means that the operating system allows different people to use the same host at the same time. Multi-tasking means that each user can run many different programs simultaneously. One of the natural functions of such operating systems is to prevent different people (or programs) using the same computer from interfering with each other. As a result, some form of security is needed. UNIX has sophisticated security mechanisms that control the way users access files, modify system databases, and use system resources. Unfortunately such mechanisms can be misconfigured, are used carelessly, or contain buggy software. The majority of the vulnerabilities that have been found in UNIX over the years have resulted from these kinds of problems rather than from shortcomings in the intrinsic design of the security mechanisms. Particularly, the following parts of UNIX have been targeted: file systems, processes, network services, X Window System and password files [9]. Despite these problems, many UNIX manufacturers still believe that they can provide a reasonably secure UNIX operating system. Many issues appear when focusing on higher security. The biggest issue is due to the habits and expectations of the end users. Many users have grown to expect UNIX to be configured in a particular way. For example, at a university level, many users are used to making their files world-readable by default. Users are also often accustomed to being able to build and install their own software, often requiring system privileges to do so. Things may change drastically as a user moves from an academic environment at a university to the corporate world. The user may carry his/her past experiences to the new environment. These expectations and habits run contrary to good security practices in the business place. As a result the system administrator needs to make a timely check on the system for the vulnerabilities and take necessary action. The first objective of this thesis is to describe a new tool named Ferret, which will help system administrators and users to make a timely check of the system and look for the problems mentioned above.

Before providing an overview of the thesis, we first review some concepts that we will be using in the thesis and that are taken from [1]. An *attack* is a malicious act that attempts to exploit a weakness in the system. Such a weakness is called a (*security*) *vulnerability*,

which is an accidental fault, or a malicious or non-malicious intentional fault. An *intrusion* results from an attack that has been (at least partially) successful. An attack is thus an intrusion attempt. An intrusion can thus be seen as the exploitation of a vulnerability. Vulnerabilities can be classified according to the location they are found. More specifically they can be found at the host or system level, network level and application level. About a decade ago the focus of security engineers shifted from host vulnerabilities to network level vulnerabilities. Security mechanisms like firewall implementations, encryption technologies are efficient in combating network level vulnerabilities. The popular doctrine was that if the perimeter or boundaries are secured then the home is safe. But despite all the perimeter defense security mechanisms, breaches have become more and more common and an estimated 70% of security breaches are committed from inside a network perimeter. Even if all the insiders are trustworthy, tools can be used to bypass most port blocking perimeter defense mechanisms such as firewalls [19]. To worsen the situation the increase in popularity of web services and Internet commerce has brought with it the usual tirade of prematurely released software littered with security holes. All these happenings and situations have again forced enterprises to reassess security from a host-based perspective.

This thesis focuses on studying host vulnerabilities. In this thesis, host vulnerabilities are considered in a broad sense. Some might argue that some of the vulnerabilities are in fact configuration errors or that some of the vulnerabilities are features provided to the user. Despite such arguments we simply use the term vulnerability for anything identified as potentially exploitable by the attackers.

There were two tools already existing that were used to check for host vulnerabilities. One of them COPS [6] was developed in 1993 by Daniel Farmer and Eugene H. Spafford. COPS [3] checks system configurations such as file and directory permissions. It also performs simple password checking. COPS relies on the fact that users and system administrators tend to be lazy about security [6]. Therefore, the assumption is that vulnerabilities in files, directories, and maybe passwords could be used to leverage access to the user's account. COPS's main strong point is that it is very easy to use and is non-intrusive (i.e., COPS does not alter files). Moreover, it reports all the problems to the auditor.

Another tool also developed about ten years ago was Tiger [16]. Tiger consists in a set of scripts that scan a UNIX system looking for security problems, in the same fashion as COPS. It scans system configuration files, file-systems, and user configuration files for possible security problems and reports them.

As said earlier these tools were developed about ten years ago and as a result have several limitations [7]. For example, they check for vulnerabilities that are no longer relevant, and offer no simple way to modify the list of checked vulnerabilities. Furthermore, they do not check for certain important current vulnerabilities. Moreover, their implementation is too customized to specific UNIX operating systems and versions. For example, they need to be modified in order to run on the current versions of Linux. Finally they were not designed using a modular approach (making updates to checked vulnerabilities difficult). For all the above reasons, we developed a new vulnerability checking tool called Ferret. The design and the implementation of Ferret are discussed in the first part of this thesis (Chapter 2).

The second part of the thesis stresses on the data analysis of host vulnerabilities. The aim of the data analysis exercise is to understand the dynamics of the evolution of vulnerabilities on a system. Vulnerabilities on a system are bound to vary from one network to another, for one type of user population to another; or talking for the same network it may vary with time also. More specifically, some vulnerabilities exist for a long period of time, some impact more users and some are removed and then re-introduced over time. Some depend on the proficiency level of the user population (e.g., users in a flexible university environment may not be compared to users on a system in some corporate office), frequency and usage of the system, feedback or action by system administrators, changes in the administrative policies, criticality of the network or the system, type of operating system, etc. For each of these factors a comprehensive research needs to be conducted first to obtain the metrics of the various factors affecting the growth of vulnerabilities and then to find the correlation between these factors and the evolution of vulnerabilities over time. Not all of these factors are independent. For example, user proficiency will increase upon feedback from the system administrator. Moreover some of these factors may either lead to proliferation of the vulnerability or may curb its growth.

The purpose of host-based vulnerability analysis can be as simple as knowing how many vulnerabilities of each type are present on a host. What types of vulnerabilities affect the highest number of users? Or finding out what vulnerabilities are predominantly common on most of the networks. Also it will be interesting to know what percentages of total users are afflicted by a particular vulnerability. For example, given 'n' users of a particular type of users' community what percentages of users are expected to have weak passwords (a weak password is defined in Section 3.3.3)? A good statistical survey of the host vulnerabilities is needed for finding answers to these questions. This information captured at a particular instant of time represents a snapshot of the system and can provide interesting results for system administrators or users. This in turn can alert and help them in their endeavor to secure the system. But for broader understanding of the dynamics of the system, the evolution of vulnerabilities needs to be observed over a period of time. For example, exploitations related to the file `.rhosts` become possible when a user has his/her username in another user's `.rhosts` file on the same system. As a result, the user can get access to the other user's account without supplying a password. It may be interesting to determine the average duration of time before a user removes the entries from his/her `.rhosts` file. Another example is to determine how long a user keeps his/her home-directory world writable before properly configuring it. Also it will be interesting to know the average duration for which a user keeps his/her password weak before making it strong.

Measures are introduced in Chapter 3 that help understanding the evolution of vulnerabilities. The majority of these measures are aimed to study the evolution of vulnerabilities and some of them are defined to observe the behavior of users also. Summing up, the process of learning the dynamic evolution of vulnerabilities for a system can be divided into three stages. In the first stage, the factors and security-related events for the system affecting the evolution of vulnerabilities need to be sorted out. The second step consists of obtaining the correlation between these factors, events and the various measures reflecting changes in the evolution of host vulnerabilities. This will enable us to know for example, how many users are going to make their password strong on getting a feedback from a system administrator (an event). The final stage will be to get the average estimations for various measures for diverse environments by analyzing

the collected data. The data used for the analysis purpose should not only be collected for a longer period of time but also over diverse networks and environments.

To estimate the measures and to study the evolution of vulnerabilities require real time data. The data are sensitive and such difficult to obtain them. Among the publicly and freely available data [2], there are many pitfalls. They are either very generic in nature (e.g., number of vulnerabilities per year) or only provide a snapshot of a system at a particular time but do not provide information about how vulnerabilities evolve over a period of time. Therefore, we have started the process of data collection on host vulnerabilities.

The thesis is divided in two parts.

a) The first part (Chapter 2) describes a new tool called Ferret developed specifically to help in identifying host-based vulnerabilities on UNIX systems and providing useful information to system administrators. Ferret has been described in [15]. The purpose of this tool can be extended to collect relevant data also to suit the second purpose of this thesis described below.

b) The second part (Chapter 3) of the thesis starts with defining measures helpful to understand the evolution of vulnerabilities for a system over a period of time. The setup for automating the data analysis process for the output of Ferret is described. Also the introduced measures are estimated using two sets of experimental data sets.

Chapter 2: Ferret: A Host Vulnerability Checking Tool

2.1 Introduction

Since the creation of the UNIX operating system in 1969 it has evolved into the cornerstone of many of the mission-critical systems or Internet servers. UNIX has sophisticated security mechanisms that control the way users access files, modify system databases and use system resources. Unfortunately, those mechanisms become less efficient when the mechanisms are misconfigured or are used carelessly. This chapter outlines an attempt to identify the above listed issues through a scanning tool called Ferret, which is a host vulnerability checking tool and is publicly and freely available.

2.2 Ferret Design Goals

In the introduction section the limitations associated with the already existing host vulnerability scanners (like Tiger [16] and COPS [6]) have been listed. The design goals of Ferret have been set up to address these limitations. Moreover, some of the design goals of Ferret are inspired by the design goals used for Nessus [12], a popular tool in its category of network vulnerability scanner. Nessus [12] is free, open-source, and based on plug-ins, and targets multiple operating systems. Ferret's design goals are as follows. Since the field of computing security changes extremely rapidly, checking vulnerabilities requires a modular approach. To this end, the framework for Ferret is designed to use independent plug-in modules, or plug-ins. Each plug-in has a specific task vulnerability to scan for. This allows quick development and deployment of new vulnerability checks that do not require modification to the management portion (or core) of Ferret. So core of the tool is platform independent. In addition, plug-ins should be capable of interacting with a variety of other vulnerability checking tools, without affecting the core.

The plug-ins are designed to be as specific to particular vulnerabilities as possible, allowing each vulnerability plug-in to check only a single vulnerability. The decision to use that approach, instead of larger multi-purpose plug-ins, made it possible to check new vulnerabilities without modifying the code for the old vulnerabilities. The keyword functionality described later can be used to group the specific vulnerabilities into larger groups of vulnerabilities. Much of the security field revolves around quick response to

new vulnerabilities and the sharing of those findings among the security community. To help with this need, Ferret has been designed as an open-source project, with the goal of creating a community of contributors that will help keep the available vulnerability plug-ins current with security discoveries. This will facilitate the rapid development of new vulnerability-checking plug-ins when new vulnerabilities are found.

Scanning a large server with many files and users can be very time-intensive. Moreover large amounts of resources are needed if the number of vulnerabilities to check for is large. Ferret is designed to be fast and efficient. It chooses performance and simplicity over complex additional features. There are many different operating systems in use today. All of these different operating systems have some unique vulnerability and some vulnerability that they share with others. The core of the Ferret system is designed to be platform-independent, and as many of the vulnerability checking plug-ins as possible are also platform-independent. In some cases, the vulnerability checked is specific to one platform, and thus the associated plug-in needs to be platform dependent. For example, the format of some system file may be different for different UNIX environments. Moreover, the framework is set up in such a way as to allow the individual vulnerability checking plug-ins to specify which operating systems they are relevant for. As a result, each plug-in can figure out on which system they are applicable. A keyword-grouping functionality is designed into the management portion (or core) of Ferret. This serves the purpose of organizing and grouping of the numerous vulnerability plug-ins. Each plug-in will have a list of keywords associated with it that will allow relevant plug-ins to be selected without requiring all of them to be run.

The output format was designed to be modular. This facilitates the generation of relevant reports from the tool using output plug-ins. These output plug-ins in turn generate a report from Ferret's raw data. This gives the flexibility in generating reports, which can range from raw text files to a centralized database using MySQL. Based on different requirements, plug-ins may have to collect different information about vulnerabilities. For each of the different customized requirements there should be a corresponding mode. Till now two modes have been implemented in Ferret. One mode indicates which vulnerabilities are present on the host. The second mode provides detailed information on

how the vulnerabilities found could be exploited and what privileges could possibly be gained on exploitation of those vulnerabilities.

Ferret has been designed in a modular way where a different plug-in module is used for vulnerability that is checked. The format of the output is also specified by different plug-in modules. As a result, Ferret is extensible, and can easily be kept up-to-date through addition of checks for new vulnerabilities as they are discovered. The modular approach also makes it easy to provide specific configurations of Ferret tailored to specific operating systems or use environments. Ferret project is open source software implemented in Perl. This will facilitate the rapid development of new vulnerability-checking plug-in modules when new vulnerabilities are found. The management component is designed to be platform independent, and as many of the vulnerability checking plug-ins modules also platform independent. In some cases, the vulnerability checked is specific to one platform, and thus the associated plug-in module needs to be platform-independent. In some cases, the vulnerability checked is specific to one platform, and thus the associated plug-in module needs to be platform-dependent. Keywords can group together different vulnerability checking plug-ins. The output plug-in modules also provide flexibility by creating reports, including raw text files. The remainder of this chapter is organized as follows. The detailed design, configuration, and implementation details of Ferret are described in the following sections.

2.3 Ferret Detailed Design

The reasons for choosing an interpreted language like Perl for implementing Ferret are as follows. Ferret will need to process various text files and Perl is known for its text processing capabilities. Perl is also available on many popular platforms. The interpreted nature of the Perl promotes simplicity of use. There is no requirement for a compilation procedure which makes it possible to scan large systems within a reasonable time. Lastly Perl is a very powerful and expressive language (e.g., its high level data structures, like hash tables, are very helpful).

2.3.1 Architecture Overview

The structure of Ferret can be divided into three parts as shown in Figure 2.1: Management (Ferret Core), the scanning agent (set of vulnerability-checking plug-ins, with individual plug-ins looking for specific vulnerabilities), and the collecting agent (set of output-plugins).

The type of interface used between the Ferret core and the plug-ins is command-line arguments. Standard command-line options are used to pass the options between the core and the plug-ins. This approach was chosen over the use of the Perl packages/modules to give the option of each plug-in to be run as a stand-alone program. This element of design made debugging very easy. Also this way a system administrator can simply run a single Ferret plug-in without interacting with the rest of the tool at all.

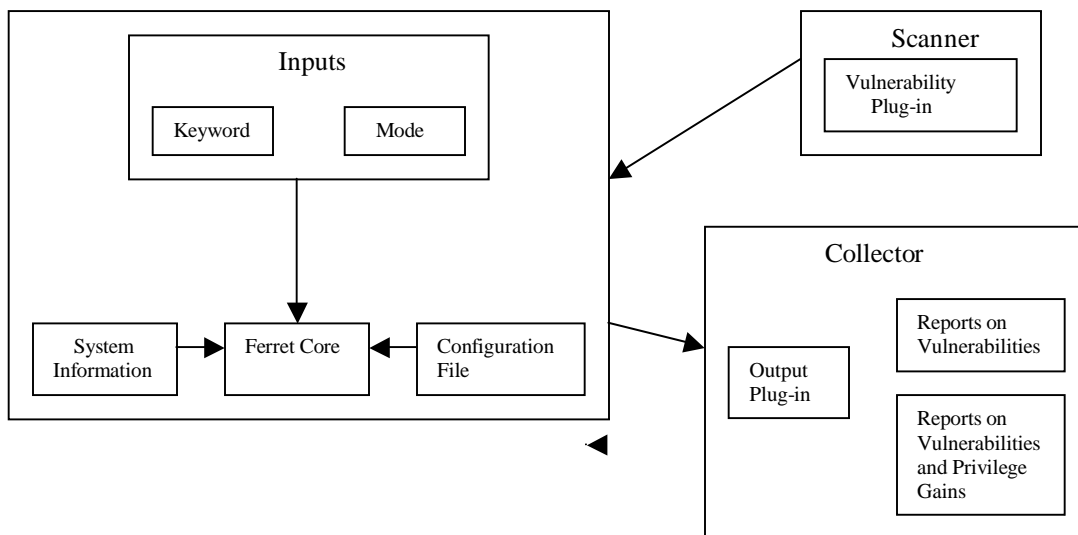


Figure 2.1: Architecture Overview of Ferret

2.3.2 Ferret Core

The core program of the Ferret tool is designed to be independent and simple. The following section details the different steps executed by the Ferret core.

1. It interprets the command line options and reads the configuration file.
2. It determines the information about the system Ferret is running on, such as the operating system name and version. This information is helpful in later stages in filtering out the plug-ins that should remain active for a particular host. For example, a plug-in developed specifically for the Linux system should not be executed on a Solaris host.
3. It determines what plug-ins are available in the system from the directory defined in the configuration file (or the command line).
4. It queries each plug-in for information about what vulnerability it checks, what level of privileges this vulnerability could gain, on which operating system this vulnerability should be checked for, and what keywords are applicable to this plug-in. This information is stored in a hash table for easy reference later.
5. It builds a hash table of the keywords, based on the individual keywords, in order to simplify the running of the plug-ins.
6. It uses the keywords to determine which plug-ins to run. If no keyword is provided, then a default keyword of all is used to run all the plug-ins.
7. It runs all the chosen plug-ins in sequence.
8. It divides the plug-ins, based on the results obtained, into two categories: those that have found vulnerabilities, and those that have not.
9. It pipes out the results, based on the mode selected by the user, through the corresponding output plug-ins in the desired format.

The source code of the Ferret core is given in the Appendix A.1.

2.4 Vulnerability –Checking Plug-ins

2.4.1 Options for Running the Vulnerability-Checking Plug-ins

The goal of vulnerability checking plug-in is to scan the system for a particular vulnerability. The Ferret core can make the plug-ins do diverse functions using the command line options. For example, when Ferret is executed with the `-info` option, each plug-in returns information including its version number, a list of keywords associated with the plug-in, the list of the operating systems on which the plug-in can be run, and a short description of the vulnerability the plug-in checks.

It is also possible for vulnerability checking plug-ins to interact with other tools. In fact Ferret plug-ins can act as a wrapper to other tools. Plug-ins written for these purposes pass the obtained results to the Ferret core. Ferret core on obtaining the results passes it over to the output plug-ins. For example, a plug-in has been written to instrument the commonly used password-cracker tool, John the Ripper [10]. This plug-in runs this tool for a specified time. This way it becomes very easy to find categories of weak passwords (defined in Section 3.3.3) on a system. Moreover the interaction with Ferret also provides how privileges can be escalated using the weak password vulnerability.

Sometimes a system administrator may want to execute only a specific subset of the plug-ins. So depending on their needs, they can enable or disable the execution of plug-ins. For example, plug-ins checking SUID files may take a lot of time because they need to search through the file system. So a system administrator running short of time can easily switch off those plug-ins.

2.4.2 Groups of Vulnerability-Checking Plug-ins

Based on the type of vulnerability a plug-in checks for, plug-ins have been divided into various groups. Till now ten groups of vulnerability checking plug-ins have been implemented. These ten groups of vulnerability checking plug-ins have been chosen based on the fact that they cover the most commonly found host vulnerabilities [8]. Vulnerabilities belonging to a particular group have similar characteristics. Currently, seventy four plug-ins have been implemented and released on the website [7] for public use. The release of Ferret as a freely available open source program should encourage the

security community to develop plug-ins and thus rapidly increase the number of available plug-ins.

The first group of plug-ins checks whether or not certain critical system files or directories (e.g., /bin, /.cshrc, /dev, /etc/group, /etc/passwd, /etc, /.login, /.profile, /.rhosts, /usr/etc) are owned by root. These files are generally run by root, and thus should be owned by root. If any ownership has been changed deliberately, then the system administrator is made aware of the change, as the aim of the Ferret is just to report a potential vulnerability and not to correct it.

The second group of plug-ins checks if the permissions of critical system directories (e.g., /, /bin, /usr/adm, /etc, /dev) are world-writable. Having any of these directories world-writable makes it easy for intruders to gain root privilege. For example, the directory /etc contains information about groups, users, and passwords. An attacker could put some commands in /etc/rc*, /etc/init.d/*, and other files in the /etc directory to create a back door into the system each time the system is rebooted. Therefore, all the files in the /etc directory should be kept non-writable by any user other than root. Similarly, the /bin directory stores shells for the system. An attacker can possibly put a Trojan Horse in any of the files to become root. It is therefore important to safeguard these critical directories. However, for specific files within the directory, Ferret leaves it up to the system administrator to decide what exactly the permission set should be. For example, although the majority of the devices listed in the /dev directory should be non-readable and non-writable, some of the files, such as /dev/null, /dev/tty and, /dev/console need to be world-writable.

The third group of plug-ins checks the paths and filenames inside the root start-up files (e.g., /.login, /.cshrc, /.profile) for world-writability. Also, the path variable in those startup files is checked to ensure that a “.” is not in the path variable. An attacker could place a Trojan Horse in various directories with the hope that someone with “.” in his or her path will execute the program.

The fourth group of plug-ins checks the umask settings in login initialization files (e.g., /.login, /.cshrc, /.profile, /etc/profile). The umask is a four-digit octal number that UNIX uses to determine the file permissions for newly created files. Umask settings often have

a default value of 022 to make it easier for users to share files with other users. Unfortunately, a user may inadvertently set the umask to be world-readable while creating a new file, which would allow anyone to read the files.

The fifth group of plug-ins focuses on configuration issues for certain files of each user's home directory. More specifically, these plug-ins check whether permissions on certain important files (e.g., `.Xdefaults`, `.Xresources`, `.bashrc`, `.cshrc`, `.defaults`, `.emacs`, `.emacsrc`, `.forward`, `.fvcmd`, `.kshrc`, `.login`, `.logout`, `.netrc`, `.profile`, `.rhosts`, `.screenrc`, `.tcshrc`, `.xinitrc`) of any user's home directory are set to group- or world-writable. This group of plug-ins can avoid many situations in which a user can snatch another user's rights easily, even by placing a simple Trojan Horse. For example, the `.login` and `.profile` files are executed each time a user logs in. Commands within these files are executed by the user's shell. Even making such files group-writable makes them susceptible to potential attackers writing commands that are executable each time the user logs in. However, to accommodate special situations in which rights or permissions have been deliberately given, the discretion of the system administrator and the user of the home directory are final.

In UNIX systems, the SUID bit allows a file to be run as the user that owns the file, even if it is run by someone other than that user. A SUID file that is writable can easily be used to compromise the user account of the owner, including the case when the owner is root. While certain files on any system need to be SUID root, it is critical that those files not be writable by anyone other than root. The sixth group of plug-ins is designed to search for all the files on the target system that have the SUID bit enabled. Plug-ins of this type find all the SUID programs on a system and check to see if they have group-writable permissions or world-writable permissions, or if they are owned by the user root (UID 0).

The seventh group of plug-ins focuses on the different fields of the password file. The structure of the password file is similar for all UNIX systems. However, certain fields may take different formats. For example, the password field may contain an asterisk (*), an exclamation (!), or encrypted password characters. Different types of vulnerabilities, like a blank password field, non-root accounts with UID set to zero, or an invalid home

directory, can cause a security flaw in the system. For example, if the password field is blank then the particular user account does not need a password to log in. Depending on the operating system and its version, the encrypted password is stored in a ghost file, like the `/etc/security/passwd` or `/etc/shadow` file. A ghost file like `/etc/shadow` gives the encrypted password a more secure structure, and such files are not readable by ordinary users.

The eighth group of plug-ins focuses on system file permissions. Some of the plug-ins developed in this family target the `/etc/exports` file, and others target the `/etc/fstab` file. More specifically, some plug-ins check whether or not files exported in `/etc/exports` have been given explicit access control. They also check whether there is any use of wildcards, or if any write access for root has been given. Note that wherever possible files, should be mounted as read-only. Other plug-ins check whether the resources that are mounted in the file `/etc/fstab` are not set as world- or group- writable. The `/etc/fstab` or `/etc/vfstab` (for Solaris) file makes it possible to automate the mounting of certain file systems, especially at system start-up. It contains a series of lines describing the filesystems, their mount points, and other options. Several plug-ins check whether the resources mounted in the `/etc/fstab` or `/etc/vfstab` (for Solaris) file are not set as world- or group- writable.

The ninth group of plug-ins checks the `.rhosts` file to see if any unreasonable permission is given to remote users and hosts. The `.rhosts` file resides in the user's home directory. The file defines which remote hosts can invoke certain commands on the local host without supplying any password. If the `.rhosts` file is writable by others, then any host included in the file can get access to run commands on the system without supplying a password. Also, if a user has his/her username in another user's `.rhosts` file on the same system, then the user can get access to the other user's account without supplying a password. Thus it becomes important to check both the content of the `.rhosts` file and the permission of the file.

Lastly the tenth group consists of two plug-ins. This group plug-ins checks if any user has set the permission on its home-directory as group or world writable.

The source code for some of the plug-ins has been attached in Appendix A.2.

2.5 Output Plug-ins

As mentioned before, the Ferret core can be run in different modes. For each of the modes, the Ferret core generates a raw format output file using the output from different vulnerability checking-plug-ins. Each mode can have a number of different formats in which to present the final output results generated by that mode. For example, a user could generate a report in XML format by defining its attributes in an XML output plug-in. As mentioned earlier, two modes have been implemented. The first mode has been designed to focus specifically on information on vulnerabilities. The second mode has been designed to provide some more detailed information on the vulnerabilities found and their possible exploitations for the purpose of quantifying the security. The source code for the output plug-in developed for the ASCII format has been given in the Appendix A.3.

2.5.1 Output Plug-in with the First Mode

The raw format output file created by Ferret core for this mode consists of four fields. The first field contains the name of the plug-in. The second field contains a short description of what the plug-in does. The third field indicates the result of the scan by the plug-in. It specifies whether vulnerability was found. The last field provides some more information relevant to the vulnerability found by the plug-in. For example, if a plug-in finds that the home directory in a password file is group-writable, then the output field will list the exact home directory that is group writable. This mode is useful to the system administrator for determining various vulnerabilities present in the system, including those in users' home directories. Based on the obtained results, an organizational security policy can be framed out, and users' negligence can be effectively checked.

2.5.2 Output Plug-in with the Second Mode

The raw format output file created by the Ferret core for this output plug-in consists of the list of all the users (along with their groups) present in the home directory of the host on which Ferret is running, plus four additional fields. The first field contains the name of the plug-in. The second field contains the information about any possible change in the privilege associated with the vulnerability. The third field provides the name of the file or directory that is vulnerable. The last field gives a brief description of the vulnerability.

The information provided by this output plug-in on the vulnerability and its exploitation can be used to evaluate the security of a computing system through the use of privilege graphs [4]. M. Dacier, Y. Deswarte, M. Kanniche have presented a method for evaluating the security of operational systems by building a privilege graph [5]. In a privilege graph [4], each node represents a set of privileges assigned to a user or set of users (e.g., a UNIX group). Any arc that connects two nodes is representative of a vulnerability, which allows a user (starting node) to achieve another user's privilege (ending node). The output file provided by Ferret contains all the different elements for building a privilege graph. The list of all the users, along with their groups contained in the output file, defines the set of nodes. The set also include two special nodes that represent a user with no privilege (an outsider) and a user who has administrative privileges. A transition between two nodes is associated with each vulnerability. The obtained output file contains the list of the vulnerabilities found on a host, and thus contains the list of transitions between two nodes in the privilege graph. The output plug-in with the second mode thus contains the textual description of a privilege graph.

2.6 Comments on the Ferret Code

The source code of Ferret given in Appendix A is well documented. But to further ease the understanding of the code more details are needed on some critical parts and are discussed here. As can be seen in Figure 2.1 the Ferret core interacts with the vulnerability checking plug-ins. More specifically, it interacts with them two times. The first time the core interacts with the plug-ins, it collects various information from each of the plug-in like the version number, the keywords that can be used for the plug-in, the operating systems for which it is meant for, the short and the long description of the plug-in and the privilege gain possible on exploiting the vulnerability. The core then stores this information in a hash table and uses it to determine which plug-ins shall be run. The Ferret core then performs various tests like checking the type and version of the operating system of the host. Only the plug-ins meant for that host operating system are run. The second time the core interacts with the vulnerability checking plug-ins it actually runs the plug-ins. As already mentioned, the Ferret core can be run in two modes. The vulnerability checking plug-ins collect information for both the modes irrespective of the mode in which the core is called. Then the information is passed over to the Ferret core,

which decides on the information to be sent over to the output plug-in based on the selected mode. This design decision was made to standardize the design of the vulnerability checking plug-ins.

All the different types of results generated by the vulnerability checking plug-ins are flushed to a single array (called “output”) and the distinction is made by the Ferret core using the keyword introduced at the beginning of a result. As an example, the plug-in bin-root-ownership given in Appendix A.2 is being discussed below.

The lines taken from the plug-in bin-root-ownership (the full code of the plug-in is given in Appendix A.2) show the different types of results pushed to the same array “output”. The keywords used help the Ferret core to distinguish the type of the result. For example, the keyword “NOT_EXIST” used below will help the core understand that the file or directory for which the plug-in was built to check the vulnerability for does not exist on that host. Similarly the keyword “RESULT” helps the core to determine if a vulnerability is found or not. And the keyword “PRIV” makes the core know the output of the plug-ins contains the change in the privilege.

```
push @output,"NOT_EXIST Directory /bin does not exist\n";
push @output,"RESULT No Vulnerability Found\n";
print @output;
die "Directory /bin does not exist\n";
push @output,"VUL None\n";

if ($vulnerability_found == 1)
{
    push @output,"RESULT Vulnerability Found\n";
}
else
{
    push @output,"RESULT No Vulnerability Found\n";
}

push @output,"PRIV No privilege change,/bin,not owned by root\n";
print @output;
```

The plug-ins can be run as standalone programs also. This helps in testing as each plug-in can be tested alone before interacting with the main core. The location of Perl on the hosts on which Ferret will be run can be changed. The first few lines at the beginning of each file in Ferret are included to find the location of Perl.

2.7 Conclusion

Ferret has been developed with the idea of providing the security community with a useful and efficient tool for checking host vulnerabilities. It is a freely available open source tool developed in Perl. The tool along with the “ReadMe” file and instructions can be downloaded from the website: <http://ferret.crhc.uiuc.edu>. Presently 74 plug-ins have been implemented. As said earlier, each plug-in checks for a specific vulnerability. Moreover, the output of Ferret is also provided through the use of plug-ins. Plug-ins developed till now mainly target Linux, Solaris, and HP-UX requirements. The security community is expected to participate in developing new plug-ins for multiple operating systems. System administrators could then select subsets of plug-in to check for vulnerabilities customized to their computer networks. Moreover, even though the vulnerabilities currently addressed focus on UNIX platforms, the development of Ferret in Perl facilitates the introduction of plug-ins targeting vulnerabilities in Windows.

Chapter 3: VULNERABILITY DATA ANALYSIS

3.1 Introduction

In Chapter 1 the importance of having various measures to study the evolution of vulnerabilities was highlighted. This chapter first introduces various measures. These measures are then estimated using data from two data collections. The first one was conducted at LAAS-CNRS over period of 21 months. The second one was obtained by running Ferret at UIUC for 9 months. Finally the results of two data collections are compared.

These measures will help understanding the evolution of vulnerabilities on a particular host over a period of time. As these measures are defined for a particular vulnerability on a system and a particular environment they are helpful in understanding the dynamics of the system under observation. The following types of questions can be answered. What is the expected percentage of users who have assigned incorrect permissions for various critical files (like `.login`, `.cshrc`) in their home-directories? How many users in the population for a given environment are expected to have weak passwords (defined in Section 3.3.3)? What is the expected duration for which a user keeps his/her password strong? Some measures can concentrate on the rate with which instances of a vulnerability in a particular network are introduced or removed. For example, what is the rate of introduction of weak passwords in a system? What is the rate of removal of weak passwords in a system? What is the average duration for which vulnerabilities stay in the system (e.g., average length for which the user keeps his/her home-directories world or group writable)? These types of estimations will help us to understand the dynamic evolution of vulnerabilities over a period of time. But it may not be easy to compare two systems based on the relative values of these measures as described afterwards when defining the measures.

Some of the vulnerabilities on a host (for a particular file system) can have only a single instance. For example, the `/bin` directory for a file system is unique so the vulnerability arising if the directory `/bin` is not owned by the root will also be unique. But some of the vulnerabilities may have multiple instances. For example, many users can simultaneously have weak passwords or may assign incorrect permission to files in their

home-directories. Each of these instances are counted as a vulnerability. As a result, some measures defined in this chapter might not be relevant for vulnerabilities having a single instance on a host. The measure rate of growth of instances of vulnerabilities on a host can be defined only for vulnerabilities having more than one instance. On the other hand, measures estimating the average duration for which a particular vulnerability is present can be useful for single and multiple instances.

For data analysis purpose the host vulnerabilities can be clubbed together in various groups based on some similarities, as shown in Table 3.1. It is much easier to study a group of similar vulnerabilities. It not only makes the study less cumbersome but it may be easier to observe any trend if studied collectively in a group. For example, all the configuration issues in `/etc/passwd` files like blank password field or a blank line can be clubbed together as a group of vulnerabilities for study purposes.

3.2 Definition of Measures

This section defines the various measures introduced to study the evolution of vulnerabilities. The measures can be divided into four categories. The first category of the measures described in the next paragraph captures the information related to the total number of instances of a vulnerability and the total number of users present on the system.

a) $T_v(t)$: This measure gives the total instances of a vulnerability v , exhibited on the system under consideration. As discussed earlier, for some vulnerabilities only a single instance is possible. This measure is not relevant for these vulnerabilities. The governing factors influencing the evolution of vulnerabilities on a system cause the estimation for this measure to vary with time. Though details on the attacks and the system is required before operational security of a system can be calculated this parameter is a reflection of the hidden potential danger a system carries with it. As a result, two different networks cannot be compared just on the basis of value of this measure. For example, it is not possible to say which network is more secure if one network has more instances of vulnerabilities than another. $T_v(t)$ is a primitive measure.

b) $U_p(t)$: Users on a system change continuously. More users may be added to a system, while some users may leave. This measure represents the total number of users on the system under consideration at time t . This measure is a primitive measure.

c) $T_v(t)/U_p(t)$: The measure $T_v(t)$ may simply be affected by the changing number of users. To mitigate that effect this measure has been defined as the ratio of total number of instances of vulnerability v to the total number of users on the system. This in effect normalizes the measure $T_v(t)$ with respect to the number of users on the system. This measure will be more useful to look at when the numbers of vulnerabilities are a direct function of number of users of the system. For example, how many users have kept their home directories group or world writeable definitely will be influenced by the number of users on the system. Here it should be pointed again that vulnerabilities having single instances may not be dependent on the number of users. Like whether `/bin` is owned by root or not is not dependent on the number of users on the system. Therefore this measure is not relevant in case of vulnerabilities having a single instance. This is a derived measure from $T_v(t)$ and $U_p(t)$.

The next category consists of three measures and captures the rate of introduction of new instances of a vulnerability in the system.

d) $N_v(t)$: On account of various reasons like a user granting permissions to everyone for his/her home directory, changing habits of user population or by just addition of new users on the system more vulnerability instances are added to a system with time. This measure indicates how many new instances of vulnerability v have been added in a system at any point of time. The advantage of this measure is that when plotted against time, a sharp rise on a particular day can be observed easily. And reasons or factors responsible for this sudden jump can be traced out. $N_v(t)$ is a primitive measure.

e) $N_{v,i}(t)$: Sometimes the fluctuations in the day to day (discrete) readings for the measure $N_v(t)$ are so high that it may not be possible to evaluate the trend of the growth of vulnerability. So it makes more sense to evaluate the change in number of instances of the vulnerability v after every fixed period of time rather than looking at discrete values (like for every day). This measure observes the growth of vulnerability after every fixed interval of time. It can be given as $N_v(t) - N_v(t-i)$ where “ i ” is any constant time interval

chosen. Since this measure can be obtained from the measure $N_v(t)$ it is a derived measure.

f) $N_{cv}(t)$: This measure counts the new instances of a vulnerability \mathbf{v} from the time of start of observation ($t = 0$) till time “t”. It can be represented as $\sum_{t=0}^{t=T} N_v(t)$. The slopes for these curves over a time period give an average estimation for the measure $N_v(t)$ and are very handy. Since this measure can be obtained from the measure $N_v(t)$ it is a derived measure.

The third category consists of three measures and captures the rate of removal of a vulnerability from the system.

g) $R_v(t)$: This measure is the counterpart to the measure $N_v(t)$ and indicates how many instances of vulnerability \mathbf{v} are removed in a system at any point of time. On account of various factors instances of vulnerabilities are also continuously removed from a system. This can happen on the account of an action by a system administrator or the increase in user’s awareness regarding the vulnerability. $R_v(t)$ is a primitive measure.

h) $R_{v,i}(t)$: On the same reasoning as given for the measure $N_{v,i}(t)$, the measure $R_{v,i}(t)$ has been defined and is the counterpart to $N_{v,i}(t)$. This measure computes how many instances of vulnerability \mathbf{v} have been removed from the system after every fixed interval of time. This measure can be given as $R_v(t) - R_v(t-i)$ where “i” is any constant time interval chosen. Since this measure can be obtained from the measure $R_v(t)$ it is a derived measure.

i) $R_{cv}(t)$: This measure is the counterpart for the measure $N_{cv}(t)$ and counts the removed instances of vulnerability \mathbf{v} from the time of start of observation ($t = 0$). It is given as $\sum_{t=0}^{t=T} R_v(t)$. The slopes for these curves over a time period give an average estimation for the measure $R_v(t)$. Since this measure can be obtained from the measure $R_v(t)$ it is a derived measure.

The last category consist of two measures and capture the cahractersitics of the user population.

j) $A_{s,v}$: This measure is defined to capture the behavior of the users of the system. It estimates the average duration of time for which a vulnerability v was present for a user over the period of data collection. It represents an average value computed over all the users. $A_{s,v}$ is a primitive measure.

k) $L_{s,v}$: This measure is also defined to help understand the behavior of the users of the system. Sometimes the user can make the system assailable with the same vulnerability intermittently in a cyclic way. So this measure represents the average time period for which the user has kept the system continuously vulnerable with a particular vulnerability. The best example can be how a user chooses a password. Sometimes a user can be careful and chooses a strong password but sometimes he/she can choose a weak password also. So this measure estimates the average time interval for which the user keeps his/her password weak before choosing a strong password. This measure depends largely on the user habits, system administrator feedbacks, etc. $L_{s,v}$ is a primitive measure.

While the first eight measures (except $U_p(t)$) are defined to study how vulnerabilities manifest themselves in the system over a period of time the last two measures along with the measure $U_p(t)$ concentrate on studying the characteristics of the user population. The average estimation for the measure $A_{s,v}$ for any two users in the population may be the same for a vulnerability v , but by looking at the estimation for the measure $L_{s,v}$, something more can be learned. For example, there may be a case when the estimation for measure $L_{s,v}$ for the weak password vulnerability is smaller for one user than the other though the estimate for the measure $A_{s,v}$ is the same. So, it can be said that this might reflect the careless attitude but on the other hand a user with larger length of sequence may reflect his/her ignorance about that vulnerability being present. After defining the measures the next step was to obtain the estimates for these measures using the two data collections one conducted at LAAS-CNRS without using Ferret and the other one conducted at UIUC using Ferret.

3.3 Data Analysis: LAAS-CNRS

3.3.1 Introduction

The data was collected at LAAS-CNRS using scripts wrapping the scanning tool COPS [3] and the password cracker tool, Cracker Version 5.5 [11] on a large distributed computer system. This system consisted of more than a hundred different workstations connected to a local area network. On an average there were 770 users during the period of study sharing one global file system. The number of users varied from 692 to around 850. The system was observed during 21 months starting from June 1995 till March 1997. The time intervals in between subsequent observations were not constant as shown in Figure 3.1.

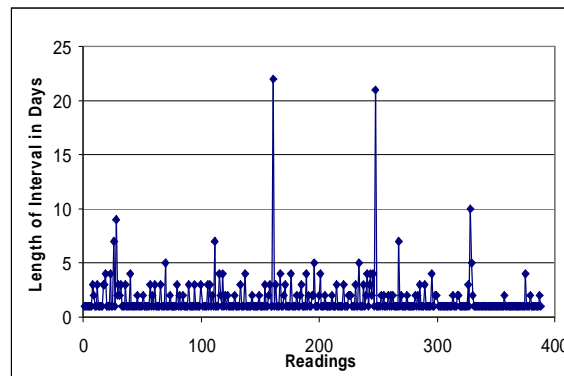


Figure 3.1: Time Intervals between Subsequent Readings

The total numbers of users on the system were continuously changing as can be seen in Figure 3.2. Throughout the period of data collection on the one hand new users kept being added to the system (Figure 3.3) and on the other hand existing users kept on leaving (Figure 3.4). This was primarily due to arrival and departure of temporary staff [13]. The archive of the data contains 385 items (one for each day). Security was not the main concern on this target system for the users. There was no strong global security policy. The number of system administrators also kept on varying from three to six during the period of data collection. The responsibility of the system administrator ranged from every day tasks like creating an account for a user, taking back-ups to specific tasks like analyzing any intrusion activity. The results of this data collection were not given to the system administrators to let evolve the system naturally. But they took some actions

based on their own network analysis. The final data presented to us was sanitized and did not represent the real users and the groups.

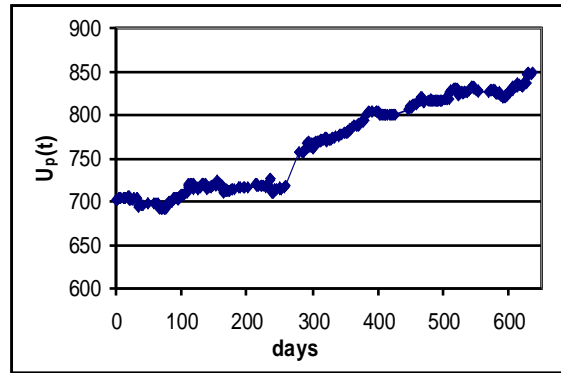


Figure 3.2: Total Number of Users as a Function of Time

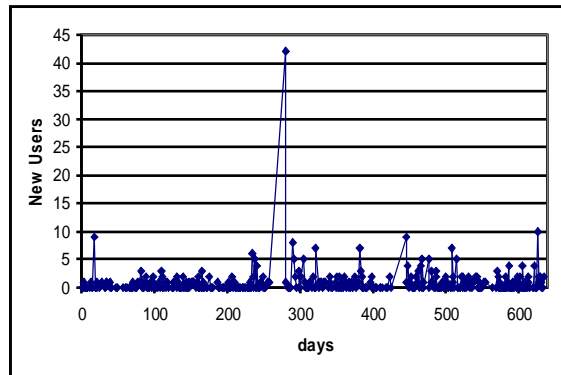


Figure 3.3a: New Users added to the System

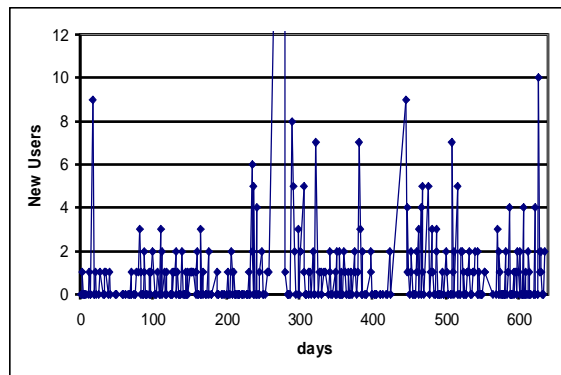


Figure 3.3b: New Users added to the System

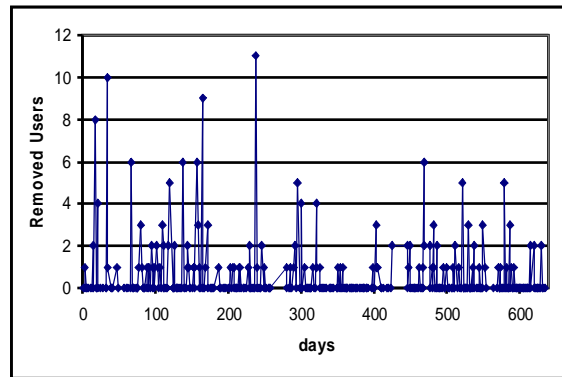


Figure 3.4: Users removed from the System

The data consisted of two types of information. The first type of information provided the users on the host along with the corresponding groups they belonged to. The second presented the vulnerabilities present and the users affected by it. After a preliminary examination of the data 31 distinct vulnerabilities were found. These vulnerabilities are listed in Table 3.1. For analysis purpose they were divided into thirteen groups. These groups were formed as to club together the similar type of vulnerabilities into one group. For example, Trojan horses can be put if either of the configuration files .login, .emacs or .profile are world or group writable. So all these vulnerabilities where a Trojan horse can be inserted on account of improper permission settings were put into one group. Those vulnerabilities, which could not be put in any group, were kept in a general category and were studied independently. For example, weak passwords could not be grouped with any other vulnerability so it was studied in isolation. Note that this classification is broad in nature. For example, all the exploits using paths were classified under “path” type vulnerability.

Out of 389 files in the archive, ten files were not used for analysis purpose on account of the following reasons:

a) The files contained only one or two vulnerabilities. So it was doubtful whether all the scans had been performed.

b) If there was a huge sudden change in number of vulnerabilities from the preceding and the succeeding recordings.

Type of Vulnerability	Exploitation possible	Group of vulnerability
A1) .login world writable	Trojan Horse	A
A2) .emacs group writable	Trojan Horse	A
A3) .logout group writable	Trojan Horse	A
A4) .tcshrc group writable	Trojan Horse	A
A5) .netrc group writable	Trojan Horse	A
A6) .xinitrc group writable	Trojan Horse	A
A7) .Xdefaults group writable	Trojan Horse	A
A8) .Xresources group writable	Trojan Horse	A
A9) .cshrc group writable	Trojan Horse	A
A10) .screenrc world writable	Trojan Horse	A
A11) .defaults world writable	Trojan Horse	A
A12) .fvcmd world writable	Trojan Horse	A
B) Password Weak	Password weak	B
C) .rhosts group writable	Group members can write in file .rhosts of the user	C
D) .mailrc group writable	Group members can make an indirect attack by .mailrc	D
E) .forward group writable	G makes a direct attack by .forward	E
F) home directory group or world writable	Group members or world can write in user home directory	F
G) User U1 in .rhosts of user U2	U1 in .rhosts of user U2	G
H1) path: /home/U/ used by User U is world writable	path /	H
H2) path: /usr/local/imsl/imsl.idf used by User U is world writable	path /	H
H3) path: /usr/local/artist/Artist2.4/bin used by User U is world writable	C. path /	H

H4) path: /usr/local/artist/Artist2.4/usrcds used by User U is world writable	C. path /	H
H5) path: /usr/local/Divers/bin/sun4 used by User U2 is writable by User U1	C. path /	H
H6) path: /usr/local/tex/bin/sun4 used by User U2 is writable by User U1	C. path /	H
H7) path: /bin used by User U {in pos. 4, 7} is world writable	C. path /	H
H8) path: /usr/local/bin used by User U {in pos. 6, 4, 10} is world writable	C. path /	H
H9) path: /usr/local/matlab/bin used by User U is world writable	C. path /	H
I1) system directory: /usr/local/bin is world writable	. system file	I
I2) system directory: /bin is world writable	. system file	I
I3) system file: /usr/local/bin/xxx used by root is writable by User U1	. system file	I
I4) system file: /etc/yyy is group writable	. system file	I

Table 3.1: Types of Vulnerabilities and their Groups

Assumptions

The assumptions made for the analysis purpose are listed as below.

- a) As already mentioned, the data was not collected every day. The time intervals in between the subsequent readings are shown in Figure 3.1. For the analysis purpose, it has been assumed that the status of vulnerability (i.e. if it is present or absent) remains the same till the next recording.
- b) The user population (though changing throughout the period of data collection) was more or less homogenous so results are not biased. This assumption can impact the estimation for measures for vulnerabilities like weak passwords. It may be expected that the users who were present throughout the period of data collection might have a better understanding of keeping their passwords strong. This might be on account of feedbacks received by system administrators or due to their own experiences of keeping a weak password.
- c) The ten files, which have been deleted as discussed before, do not significantly affect the results.

Limitations of the Study

- a) As already mentioned the LAAS-CNRS data was collected more than 7 years ago. So the estimation they provide for the measures may not reflect today's scenario.
- b) The system administrators' policies are also not known in details. It is not known what inputs were given to the users and how forceful they were. Because of these limitations in the data analysis, trends can be observed but they can't be pin-pointed to a particular factor.

3.3.2 Working Methodology

The procedure is being outlined here in form of various steps. The first step was done for every group (Table 3.1) of vulnerability but the other steps were repeated for only five groups for reasons discussed afterwards.

- 1) First a generic trend of the behavior of the total instances of the vulnerabilities with time ($T_v(t)$) was obtained for each group of vulnerability. Sometimes to obtain more specific information analysis was then narrowed down to an individual vulnerability in the group. This was done particularly to see what vulnerabilities in the group have a dominant influence on measure $T_v(t)$ and to see if the behavior of the individual vulnerabilities of the group is the same with the group as a whole.
- 2) To make the obtained trends not biased on the number of users' plots of measure $T_v(t)/U_p(t)$ were obtained. The plots obtained in step 1 and step 2 together with users' curves (Figures 3.2, 3.3, 3.4) were compared to see how changing user population influences the behavior of the total number of instances of the vulnerability.
- 3) To study the jumps in the above two plots obtained, measure $N_v(t)$ plotted against time was obtained and reasons to explain these sudden jumps were sought out.
- 4) The plots for measure $N_{v,i}(t)$ are then obtained. Different time intervals were empirically tried out to choose the best interval, "i" which helps to understand any trend in new instances for the vulnerability.
- 5) The plots of $N_{cv}(t)$ were obtained against time. The slopes for these curves over a time period give an average estimation for measure $N_v(t)$.

- 6) The plots of $R_v(t)$ over the period of data collection were obtained. The jumps are studied and the corresponding reasons were sought out.
- 7) The plots for measure $R_{v,i}(t)$ are then obtained. Different time intervals were empirically tried out to choose the best interval “ τ ” to show the trend in the data.
- 8) The plots of $R_{cv}(t)$ were obtained against time. The slopes for these curves over a time period give an average estimation for measure $R_v(t)$.
- 9) Finally curves showing the net rate of change in the instances for the vulnerability were obtained. The net rate can be given as $(N_v(t)-R_v(t))$. This rate is an approximation to the first derivative of the function $T_v(t)$. Similarly curves were obtained for $(N_{v,i}(t)-R_{v,i}(t))$ and $(N_{cv}(t)-R_{cv}(t))$ and plotted against time. Sometimes the rate of introduction of new instances $N_v(t)$ and rate of removal of instances $R_v(t)$ cancel each other for long time periods. This makes the measure $T_v(t)$ constant with time and makes the system appear static. But the reality may be that instances of vulnerabilities are both added and removed continuously from the system.
- 10) Finally estimations for the two measures $A_{I_{s,v}}$ and $L_{s,v}$ were computed.

3.3.3 Results and Observations

Out of the nine groups shown in Table 3.1, only five groups (i.e. groups A, B, D, F and G) were studied in detail after step one of the working methodology. Basically step one along with some preliminary examination of the data helped in reaching this conclusion. The prime reason for dropping the other groups and focusing on only five groups was that the instances of the vulnerabilities for the dropped groups were very small and insignificant compared to the chosen five groups. Besides this the count of their instances over a period of time was constant.

Table 3.2 lists the groups that were studied comprehensively along with the corresponding estimation of the measure $A_{I_{s,v}}$ and $L_{s,v}$ calculated for the users.

Group	$A_{s,v}$ (days)	$L_{s,v}$ (days)
A	260	209
B	168	154
D	249	191
F	137	80
G	331	240

Table 3.2: Estimations for the Measures $A_{s,v}$ and $L_{s,v}$

Results for Group A

The results presented in this section focus on vulnerabilities in configuration files. The vulnerabilities are due to incorrectly assigned permission on these files. As a result, the vulnerabilities can be exploited by inserting a Trojan horse in the configuration files. Table 3.1 shows a total of twelve vulnerabilities found due to improperly assigned permission settings on the following twelve files: `.login`, `.emacs`, `.logout`, `.tcshrc`, `.netrc`, `.xinitrc`, `.Xdefaults`, `.Xresources`, `.cshrc`, `.screenrc`, `.defaults` and `.fvcmd`.

As can be seen in Table 3.1, Group A consisted of a total of twelve types of vulnerabilities. First a plot for measure $T_v(t)$ against the number of days for the whole group was obtained (Figure 3.5). Three distinct regions can be seen in the graph based on the value for this measure. In the first region ranging till 138th day, the value for measure $T_v(t)$ varied from 232 to 281 and increased steadily. As can be seen there is a sudden jump on day 138 for measure $N_v(t)$ (Appendix B.1). This observation cannot be explained based on the information we have. Also the sudden dip on day 166 and rise on day 169 cannot be explained. The data have been cross-checked to be sure of these observations.

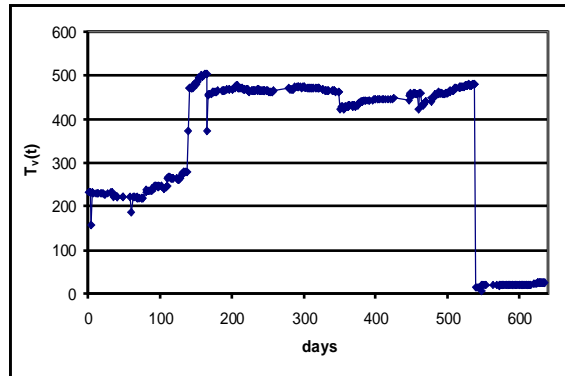


Figure 3.5: $T_v(t)$ for Group A

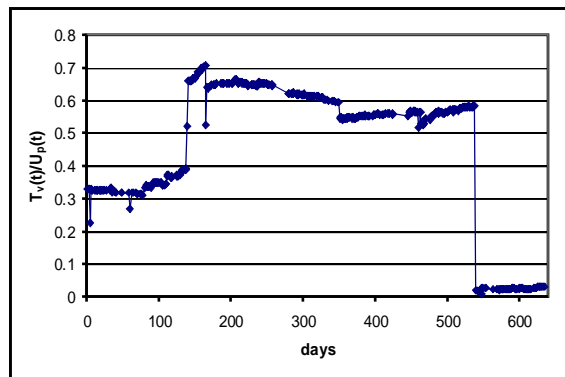


Figure 3.6: $T_v(t)/U_p(t)$ for Group A

The plot for measure $T_v(t)/U_p(t)$ (Figure 3.6) also showed three regions as expected. At one point of time the ratio was as high as 0.7. But due to the fact that a user can simultaneously set incorrect permission on two or more files in his/her home-directory this ratio does not indicate percentages of users of the population having this vulnerability. The second region started from day 140 to 538 and is quite stable. Then suddenly the value for measure $T_v(t)$ suddenly drops. This huge drop in the number of instances for the vulnerability is due to the conjunction of the action of the system administrators for removing this vulnerability. The third region was more controlled initially but then again the number raised. This important increase over the last phase is also difficult to explain. It may be due to new users added to the system who would

implement the similar type of configuration error. We then focused in more details on some more specific vulnerabilities. Were all the different vulnerabilities similar in nature of their behavior to the aggregate of them? Measure $T_v(t)$ was plotted against time for the group members. The main contributors for the total number of instances were the vulnerabilities A1, A2, A4, A6, and A7 and showed the similar trends and behavior as the group. The instances for the other vulnerabilities in Group A were insignificant in number to influence the behavior. Numerically they all were within range of one to five instances. But measure $L_{s,v}$ was very high for all of them. This shows that these very dangerous vulnerabilities were hiding in the system for long before they were taken care of. Moreover the estimation for measure $A_{s,v}$ and $L_{s,v}$ have been nearly equal for them. Another strange thing observed was the revival of the vulnerability A11 on day 538. When all the vulnerabilities were removed from the system this vulnerability surfaced and lasted till the end of the data collection. The average numbers of days the vulnerabilities of this group were present for the users were 260 days. The length of days for which this vulnerability was present continuously without any break was 209 days.

Results for Group B

The results in this section detail the number of weak passwords observed during the 21 months period at LAAS. A password is defined as weak if it would be short or which could be rapidly guessed by searching a subset such as words in the dictionary, proper names, words based on the user name or common variations on these themes [14]. On the other hand, a strong password would be sufficiently long, random, or produceable only by the user who chose it. On guessing attacks, it can be said that on an average it will require considerably more time than a weak password. There are three variables for a password cracker tool that can be set: time for which it is run, the size of the dictionary it uses and rules applied on the dictionary words. A weak password is defined with respect to these three parameters. For example, given a fixed set of rules and a dictionary a password will be considered as weak if it is cracked in less than one hour. For the data collection at LAAS-CNRS the definition of a weak password was fixed throughout the period of data collection.

Figure 3.7 shows the plot of the measure $T_v(t)$ for weak passwords. Figure 3.8 shows the plot for measure $T_v(t)/U_p(t)$. Since each user had only one user account this measure also represents the percentages of users who had weak passwords. For the first figure we observe that there is a sudden jump in the increase of weak passwords on day 283. The numbers of weak passwords increase from 107 to 120. This is probably due to the large number of new users introduced on that day (around 42) as can be seen in Figure 3.3a. In general for Figure 3.7, we can see a steady rise in the number of weak passwords until around day 480. Then the number of weak passwords drops significantly. This huge decrease is due to an action from the system administrators contacting the users with weak password urging them to modify them. Still 8% of the users continued having weak passwords.

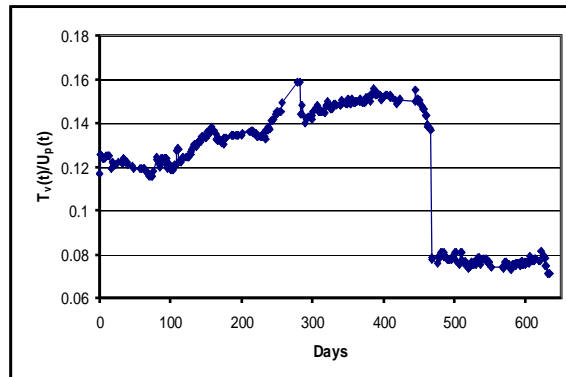


Figure 3.7: $T_v(t)$ for Group B

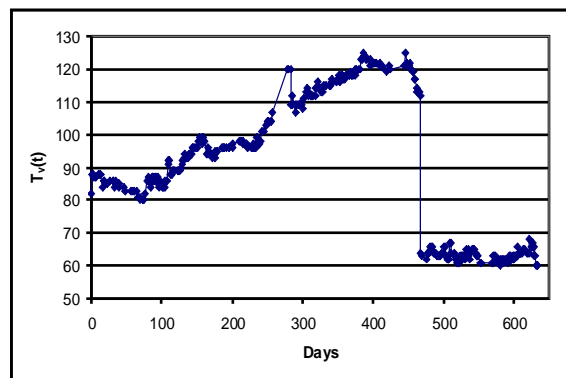


Figure 3.8: $T_v(t)/U_p(t)$ for Group B

The plots for the measures $N_{cv}(t)$ and $R_{cv}(t)$ are shown in the Figure 3.9 and 3.10 respectively. The sudden increase in the plot for $R_{cv}(t)$ around day 480 is due to the action

by the system administrator as mentioned before. The jump in both the graphs around day 300 is unexplained. One important conclusion driven from these graphs is that both the rate of introduction of new weak passwords $N_v(t)$ and rate of removal of weak passwords $R_v(t)$ in the system remain constant for different periods. Even after the action of the system administrators this observation holds true. The estimation for $N_v(t)$ till day 282 is 0.41 and becomes 0.40 after day 287. The estimation for $R_v(t)$ for different time periods are: 0.28 before day 284, 0.41 from day 285 to day 468 and 0.42 after day 469. The mean value estimation for the measure A_{1s} for this group of vulnerabilities was 168 days and 154 days for L_s . The last result implies that the mean time for which a user kept his/her password weak continuously before making it strong is around 154 days.

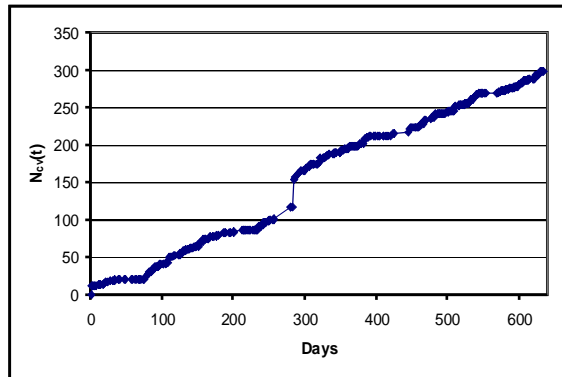


Figure 3.9: $N_{cv}(t)$ for Group B

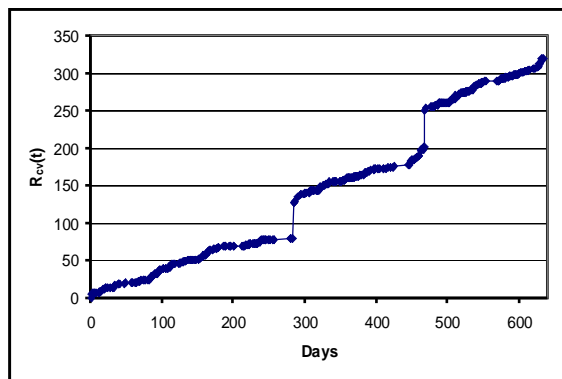


Figure 3.10: $R_{cv}(t)$ for Group B

Results for Group D

Figure 3.11 and Figure 3.12 present the plot of measure $T_v(t)$ and $T_v(t)/U_p(t)$ for a group D vulnerability. In effect the first graph shows the number of files .mailrc that contains a vulnerability. The exploitation of this vulnerability allows an attacker to obtain super user privileges. Looking at these figures several distinct phases can be identified. The first phase until around day 80 corresponds to stable number of vulnerabilities. The second phase from day 80 to day 200 witnesses a sharp increase in $T_v(t)$. From day 136 to 144 a sharp rise in measure $T_v(t)$ can be seen (from 22 to 42). The users who were impacted with this vulnerability during this time remained vulnerable till the end of study. $L_{s,v}$ is over 400 days for these users. From day 200 till day 530 the measure $T_v(t)$ is stable. A huge decrease appears around day 530. This decrease is probably due to an action from the system administrator to remove this vulnerability. After this decrease, a relatively stable phase appears again.

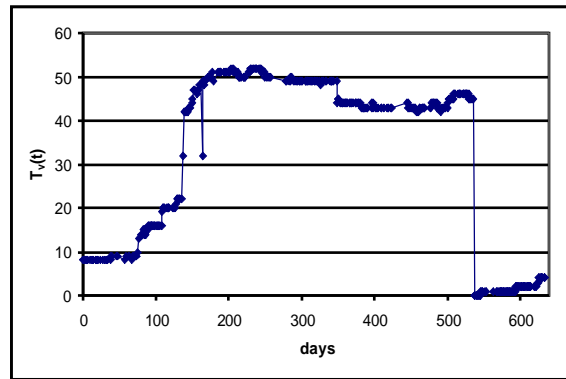


Figure 3.11: $T_v(t)$ for Group D

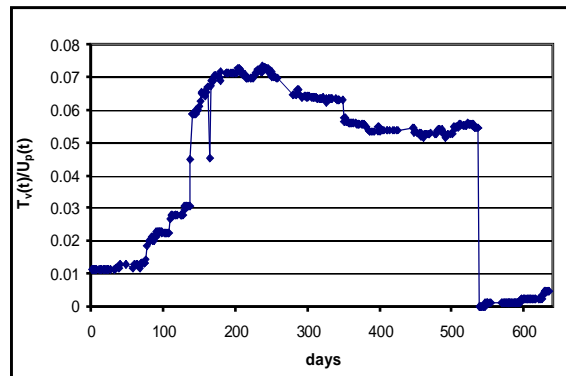


Figure 3.12: $T_v(t)/U_p(t)$ for Group D

While the important decrease in the number of vulnerabilities around day 530 is explainable, the important increase of around 120 days followed by a long period of stability is more difficult to interpret. Similarly the sudden dip observed on day 166 also cannot be explained based on the information provided, though the data has been checked carefully to confirm this observation.

Results for Group F

There are four periods that can be observed in the evolution of this vulnerability. For the first hundred days, the observed instances of this vulnerability varied from 200 to 231. One particularly dangerous observation during this time was that on the 68th day two users made their home-directory world writable thus making their directories vulnerable to all 700 users present at that time. But on day 102 these two users again assigned correct permissions to their home-directories. This was the only time during the study that anyone had made their home-directories world writable. After the first 100 days the number of this vulnerability kept decreasing and the second distinct period can be seen to exist till day 162. The number of vulnerabilities during this period ranged from 175 to 200, but they were always decreasing. After that stable periods can be seen where measure $T_v(t)$ ranged from 127 to 142. Finally system administrator took some action around day 536 and this brought down the number to just 14 from 127.

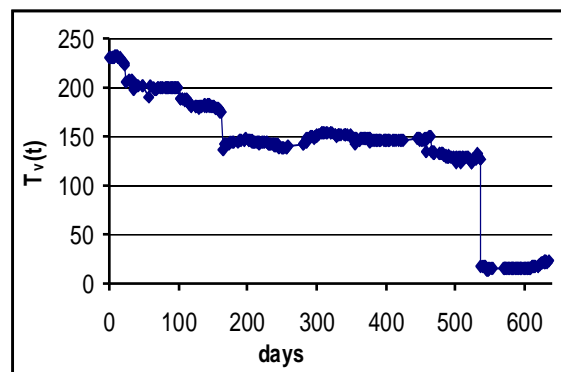


Figure 3.13: $T_v(t)$ for Group F

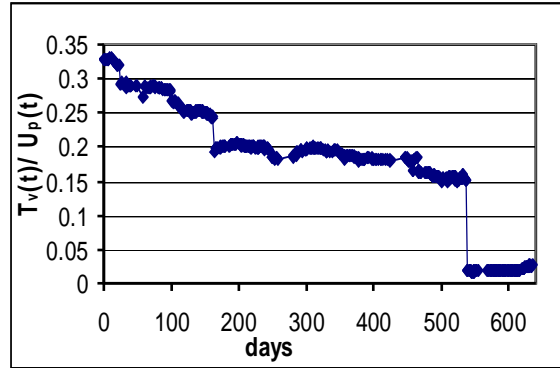


Figure 3.14: $T_v(t)/U_p(t)$ for Group F

Results for Group G

The measure $T_v(t)$ for this group of vulnerability represents the number of users sharing some of their privileges with another user through the `.rhosts` file. This sharing can be seen as a configuration feature or as a vulnerability. Instances of the vulnerability are counted for each user added to in the `.rhosts` file of a different user. Figure 3.15 shows a very stable estimation for measure $T_v(t)$. The average estimation for measure $T_v(t)$ is also 26, very low compared to other groups. Figure 3.16 shows another very stable curve for measure $T_v(t)/U_p(t)$. The value of measure $T_v(t)/U_p(t)$ lies between 0.025 to 0.045, the most stable observed among all the groups of vulnerabilities studied. The average of the estimation for this measure is .034 but due to the way the vulnerability is being counted this estimation does not reflect the percentages of users afflicted by this vulnerability. The stability of the two measures discussed above are ascertained by the plots of $N_v(t)$ (users adding new users in their `.rhosts` file) and $R_v(t)$ (users removing other users from their `.rhosts` file). Both of these figures are remarkably similar leading to the stability of measure $T_v(t)/U_p(t)$ over the period of study. The sudden dip and rise observed on day 546 was due to the fact that on that day two users removed all the users present on their `.rhosts` file and the next day they again included all those users.

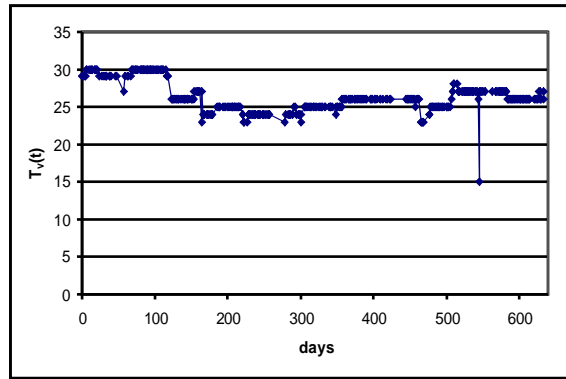


Figure 3.15: $T_v(t)$ for Group G

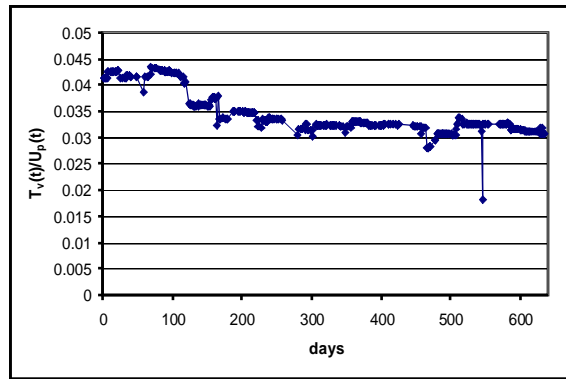


Figure 3.16: $T_v(t)/U_p(t)$ for Group G

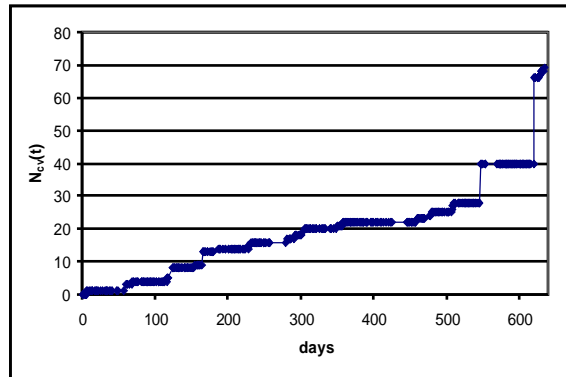


Figure 3.17: $N_{cv}(t)$ for Group G

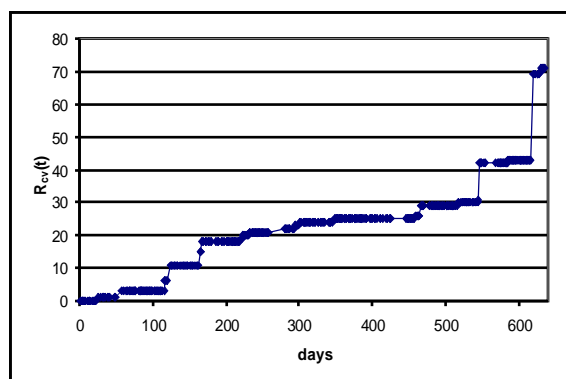


Figure 3.18: $R_{cv}(t)$ for Group G

3.3.4) Conclusions

The measure $T_v(t)/U_p(t)$ does not always represent the percentages of users afflicted by a particular vulnerability. It depends on two things: the way instances of vulnerabilities are counted and the way the groups are formed. For example, for the .rhosts vulnerability, a user can put several users in his/her .rhosts file and each listing is being counted as a separate vulnerability. The first group of vulnerabilities (Group A) can be given as another example where different files are grouped together for analysis purpose, so a user may assign incorrect permissions simultaneously to different files.

The results show that the system for which data was collected was dynamic in nature and the action of the system administrator, changing user population drastically affected the total instances of the vulnerabilities in the system. For group A, the trends observed for the following files were similar in nature: .login, .emacs, .tcshrc, .xinitrc, .Xdefaults. For group B, the most significant result observed was that both the rate of introduction of new weak passwords and rate of removal of weak passwords remained constant. For group D, it was interesting to see that 20 users had kept the vulnerability related to .mailrc files continuously for 400 days showing a total lack of awareness. For group F, the most noticeable event observed was that on day 64 two users made their home-directories world writable. This in turn made it vulnerable to all the 700 users present on the system during that time. Also estimations for measure lack of awareness. For group F, the most noticeable event observed was that on day 64 when two users made their home-directories world writable. This in turn made it vulnerable to all the 700 users present on the system during that time. Also estimations for measure $T_v(t)$ and measure $T_v(t)/U_p(t)$

consistently decreased throughout the period of data collection. For group G, the consistency observed for measure $T_v(t)$ was noticeable. At the end of the data collection when the number of vulnerabilities in other groups significantly decreased this vulnerability was not affected. After having analyzed the data collected at LAAS-CNRS, we now analyze the data collected by Ferret at UIUC.

3.4 Data Analysis: UIUC

3.4.1 Introduction

Ferret has been installed on eight hosts at University of Illinois at Urbana-Champaign to start the process of data collection. Ferret was run periodically as a cron job and data have been collected three days a week. Ferret was run in both the modes. But the results were not reported to the system administrator. The brief description of the hosts on which Ferret is run is listed in Table 3.4. The hosts are chosen as to represent different operating systems and versions, including three server-class hosts and three workstations. More specifically there are five Sun hosts, two HP-UX hosts and one Linux host. The users mainly consisted of researchers and students with strong background in computer science

3.4.2 Working Methodology

The whole process of data analysis needed to be automated because of the amount of data Ferret generates for each host. With Ferret being deployed at more and more places, the volume of data sent every week for analysis purpose will be huge. This requires an environment to be set up that makes the process of analysis easy, fast and reliable. For this purpose scripts were written in Perl DBI (Database Interface) to feed the raw Ferret data to a MySQL database and query the database. DBI allows writing Perl code that accesses data without needing to worry about database or platform specific issues. So the output of Ferret can be analyzed in the Windows environment to later take the advantage of MS-Excel data processing capabilities. One of the Perl scripts used for transferring the data to MySQL database is given in Appendix A4. This script is written for the format of data furnished when Ferret runs in the second mode.

The steps followed by the script given in Appendix A4 are given below:

- 1) Asks the user to specify the folder where Ferret output data is located.
- 2) Opens the folder and reads all the output files in an array.
- 3) Connects to the Mysql database using user name and password of the user.
- 4) Creates separate table for each corresponding file contained in the output folder.
- 5) Reads the output files one by one and writes the data to the tables in the desired format. Repeats these steps for all the output files.
- 6) Disconnects the database.

The Mysql table created by the above script contained the following fields:

Field	Type	Null	Key	Default	Extra
Id	int(11)		PRI	NULL	auto_increment
Plugin_name	varchar(40)	YES		NULL	
change_in_privilege	varchar(40)	YES		NULL	
file_affected	varchar(40)	YES		NULL	
Description	varchar(40)	YES		NULL	
Timestamp	varchar(40)	YES		NULL	

Table 3.3: Fields in the Mysql Table

Once the tables are created the next step was to retrieve the information from the table. Again the choice was Perl and Perl DBI. A couple of SQL queries were written for the purpose. A sample script is given in the Appendix A4. The steps followed by the script are given below:

- 1) Asks the user to specify the database to be queried.
- 2) Puts all the vulnerabilities in an array for which query is required to be run.
- 3) Prepares and executes the statement to create the result table.
- 4) Prepares and executes the query and puts the obtained results in the result table.

The result table generated by the above script is then exported to MS-Excel using a tool from the Intelligent Converters known as sql2xls.exe. Once the results are on the Excel

spreadsheets, VB Macros took over the task of automating various computing tasks which included calculating measure A_{ls} , plotting the measure $T_v(t)$ as a function of time for different vulnerabilities. A sample script for one of the VB Macro is given in Appendix A4. This Macro plots measure $T_v(t)$ as a function of time. This Macro is to be run in the worksheet containing the result table. The steps followed by this Macro are as follows:

- 1) Positions the chart using column and row height units.
- 2) Names the chart and sets the type of chart.
- 3) Adds data series and adds axes.
- 4) Formats the titles of the axis as desired.

3.4.3 Results and Observations

The results obtained for data collected at UIUC starting from 17th October 2003 to 27th June 2004 is discussed in the next paragraph. Table 3.4 lists all the hosts and the total number of vulnerabilities and instances. The type of vulnerabilities and the number of instances remained constant for the complete duration of studies. All the vulnerabilities that were present at the beginning of data collection remained present throughout the entire period of data collection. No new vulnerabilities appeared and as a result, the estimations for the measures $N_v(t)$ and $R_v(t)$ remained zero throughout the period of data collection. The estimations for measure $A_{ls,v}$ and $L_{s,v}$ also could not be given as the period of data collection is small. The only non-trivial estimation that can be carried out was for the measure $T_v(t)$ and is given in Table 3.5 for different hosts and vulnerabilities groups. It can be seen from Table 3.4 that host 1 contained a large number of vulnerabilities. The vulnerabilities found on this host were predominantly detected by first group (non-root-ownership of critical files and directories), fifth group (configuration issues in certain files of users' home-directories) and seventh group (vulnerabilities related to the password file) of Ferret plug-ins.

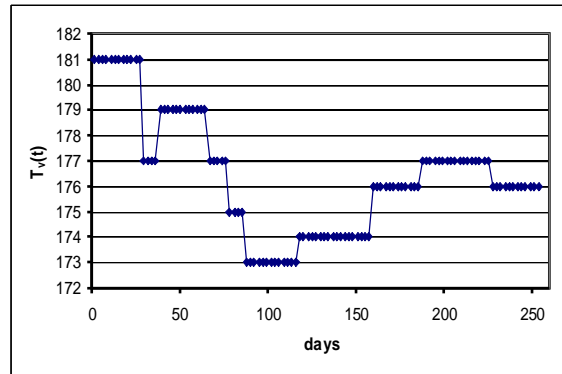


Figure 3.19: $T_v(t)$ for Fifth group

More specifically, critical directories like `/bin`, `/dev` were not owned by root. Many files in users' home-directories had configuration issues. For example, many users had set the permission on their `.cshrc`, `.login`, `.profile`, `.rhosts`, `.emacs`, `.forward`, `.logout`, `.tcshrc`, `.xinitrc`, `.kshrc`, `.netrc`, `.screenrc` files as group and world writable. The plot for measure $T_v(t)$ for this group of vulnerabilities generated using the Macro is shown in Figure 3.19. This is the only group of vulnerabilities that showed variation with time. Some of the entries in the `/etc/group` file had blank password fields, contained group names that were not alphanumeric. Moreover the file `/etc/shadow` for this host does not exist. Four users had put the permissions on their home-directories as world writable and seventeen users had put it group writable.

The rest of the hosts mainly showed vulnerabilities that were also seen on host A. Some of the other vulnerabilities were detected by the following Ferret plug-ins: `homedir-world` and `group writable`, `rhosts-username`, `set-umask-etc-profile`.

Host name	OS type	Types of vulnerabilities found	Total instances of vulnerabilities found
1	HP-UX USEOSVERSION B.11.11	33	285
2	SunOS USEOSVERSION 5.9	5	33
3	SunOS USEOSVERSION 5.7	2	18
4	Linux USEOSVERSION 2.4.20- 20.9	1	3
5	SunOS USEOSVERSION 5.9	2	20
6	SunOS USEOSVERSION 5.9	2	23
7	SunOS USEOSVERSION 5.8	3	24
8	SunOS USEOSVERSION 5.8	3	30

Table 3.4: Hosts and number of Vulnerabilities Found at UIUC

Host Name	First Group	Fourth Group	Fifth Group	Seventh Group	Ninth Group	Tenth Group
1	4	1	175*	70	20	15
2	0	1	9	16	3	4
3	0	0	0	15	0	3
4	0	0	0	0	0	3
5	0	0	0	16	0	4
6	0	0	0	18	0	5
7	0	0	4	15	0	5
8	0	1	12	15	0	2

Table 3.5: Breakdown of Vulnerabilities Found on Various Hosts at UIUC

*Average value provided for this measure

3.4.4 Conclusions

The results obtained for UIUC data are very stagnant in nature. Moreover the types of vulnerabilities present on different hosts are mainly from two or three groups of Ferret plug-ins. May be the period of data collection is too short to see the dynamic behavior. For example, the users on these hosts may not have changed.

But still it is interesting to see the number of vulnerabilities on host A. It is a HP-UX server host. One explanation behind this observation could be that this host is relatively old (about 3.5 years old) compared to the other hosts. Another reason could be the large number of users it is serving (i.e. about 335 users). The vulnerabilities found by plug-ins focusing on the non-ownership by root of critical files/directories and on vulnerabilities in the password files are serious in nature and demand more alertness on the part of the system administrator. On the other hand, the significant number of vulnerabilities found related to the configuration issues in the home directory files shows negligence on the part of users. The total number of home-directories related vulnerabilities present on host A was simply too high. Most of the vulnerabilities found are easily correctable, and a

system administrator with a tool such as Ferret can use it to check the hosts quickly (with an average time of around 120 seconds) and efficiently on a regular basis.

3.5 Comparison of the Two Data Analysis

It is interesting to compare the results obtained at LAAS-CNRS and UIUC. These two data collections were carried out for different types of user population. While at UIUC the users were mainly with computer science background, at LAAS-CNRS users consisted of users with diverse backgrounds. Not only there is a gap of 7 years between the two data collection activities but also they are at geographically distant places. The system administrators at UIUC corrected some serious vulnerabilities on seeing the first results of Ferret. But data collection started only after that phase. As an example, on noticing that Ferret output showed some users had blank password field the system administrator immediately notified the users and got it corrected. But once the data collection started the system administrator was no longer in the loop. For the comparison purpose, it should be pointed out that it may not be much significance in comparing the absolute values obtained for various measures. For example, it may not be much importance in comparing that 15% of users at LAAS had a particular vulnerability while only 10% of users were afflicted with the same vulnerability at UIUC. This comparison can only be made if there is a significant difference in the percentages of users affected by the vulnerability. Moreover because of the nature of the results obtained at UIUC most of the measures had their values as zero. So under these circumstances, the main aim of comparison can still be to see how measure $T_v(t)$ behaved for the two places.

The group of vulnerabilities that were common for both the studies are as follows:

- a) Both group A of LAAS-CNRS and fifth group of Ferret plug-ins look for vulnerabilities resulting due to configuration issues for certain files in users' home-directories. Ferret list of plug-ins is more comprehensive so the results obtained at UIUC take into account more files. But it can be seen that the results obtained at LAAS-CNRS are more dynamic in nature. Whereas there are huge jumps and falls for the estimation of measure $T_v(t)$ at LAAS-CNRS, results at UIUC are more stable.

- b) Group F of LAAS-CNRS and tenth group of Ferret plug-ins look for the same purpose of checking if any user has put permission his/her home-directory as world or group writable. Whereas at LAAS-CNRS, two users made their home-directories world writable, no such event was observed at UIUC. $T_v(t)$ was always decreasing at LAAS-CNRS whereas it was a constant function observed at UIUC.
- c) Both Group G of LAAS-CNRS and ninth group of Ferret plug-ins look for if a user has put anyone in his/her .rhosts file. The evolution of the measure $T_v(t)$ for the two data collections was similar. More specifically, at LAAS-CNRS the estimation for measure $T_v(t)$ did not show any of the dramatic changes observed for the other groups, it was constantly the same at UIUC for the nine months of data collection.

The rest of groups were exclusive for both the studies.

For the common groups it was observed that results from UIUC are very stable in nature but LAAS-CNRS results were showing a lot of fluctuations. This result is significant though absolute numbers of vulnerabilities found in both the networks may not be of much practical importance. However it is interesting to see this difference of nature in the results collected at different university environments. The change in the user population at UIUC was not that high as observed at LAAS-CNRS.

Chapter 4: Conclusions and Future Work

Ferret, the new host vulnerability checking tool described in the first part of this thesis can be run in two modes. One mode is designed to be helpful for the system administrators and the other for security community. For the system administrators it is a tool that saves a lot of manual efforts. According to system administrators' time and resources constraints only a handful of plug-ins can be picked and chosen to run. The keywords set that can be given to run a particular plug-in are very comprehensive in nature. All the possible variation of the key words that can be used to run plug-ins have been included in the plug-ins. For example, a user wanting to check if a particular directory or file is set group writable can use both the spellings: writable and writeable as keyword. Ferret second mode actually builds privilege graphs [4] that will help security community efforts to quantify the computer security [13]. The required information for the privilege graphs are collected by the plug-ins module and passed to the core of the Ferret. Thus the core of Ferret has the ability and flexibility to adopt for any new mode that may be thought in future.

The plug-ins implemented till now in Ferret have been extensively tested for different operating systems and by different people. The team included many system administrators whose valuable feedbacks and suggestions have been duly taken care of. The code is well documented, clear and precise. This will help people in developing new plug-ins and to change the existing ones according to their own requirements. Some of the critical parts of the code have been described in the thesis. The clarity of the code brings a confidence among the system administrator that the tool is not doing anything other than the intended purpose. The core and the plug-ins along with the configuration files in the tool have been tested not to use any command that delete a file or kills a process. Plug-ins developed till now mainly target Linux, Solaris, and HP-UX requirements. The security community is expected to participate in developing new plug-ins modules for multiple operating systems. Since Ferret project has been implemented in Perl there should be no problem for anyone to develop plug-ins modules targeting vulnerabilities in Windows also.

The second part of the thesis showed how a set up has been developed to automatically process the data collected by Ferret. The type of data collected by Ferret is rare. And will definitely help the security community towards the quantification of security. The type of analysis of host vulnerabilities is unique. Some more measures may need to be defined to capture the effects of various influences and factors on the measures. These measures are not only time dependent but depend on a lot of other factors. After comprehensive studies only distribution curves of various vulnerabilities will be obtained. One of the very interesting results for LAAS-CNRS data seen that was observed was that both the rate of introduction and rate of removal of weak passwords in a system was constant and not depended on time. More studies need to be conducted in these efforts before generalizing this but it is certainly an interesting result for the evolution of this vulnerability. For LAAS-CNRS the estimation for the measures A_{1s} , and L_s have been very high. It may be really a worrisome result for system administrator to see that an average user keeps his/her password weak continuously for 154 days before making it strong and for 40 days continuously two users had kept their home-directory world writable. Overall the results obtained at LAAS-CNRS were marked by huge jumps and falls in sharp contrast of what is observed at UIUC. The results obtained for UIUC showed a consistent behavior and less percentages of users were found vulnerable to attacks for the vulnerabilities checked. This result is also very significant in itself. The exact reasons behind these differences still need to be investigated and more data need to be collected over diverse population and environments. This will lead to finding out what types of possible behavior are there and what set one user population in a particular environment different from the other. In this way studies discussed in this thesis may be considered as at initial stages. The measure $T_v(t)$ need to be normalized not only with respect to number of users in the population but also with other factors. This will make the measure independent of various parameters on which it depend which will allow data to be compared across different networks and for diverse population. The curves representing the distribution of vulnerabilities for different variables need to be obtained. It is hoped that these results will motivate people to share data on computer security.

Appendixes

These appendixes provide code for the Ferret core and vulnerability checking plug-ins, graphs for the data analysis conducted on the LAAS-CNRS data, and code written for automating the data analysis process for data collected using Ferret.

Appendix A.1: Source code of Ferret Core

The source code of Ferret core is given below

```
eval '(exit $?0)' && eval 'exec perl -S $0 ${1+"$@"}'
  & eval 'exec perl -S $0 $argv:q'
  if 0;

#Copyright 2002, 2003, 2004, University of Maryland and University of
#Illinois. All rights reserved.

#Developed by: Jason Martin, Anil Sharma, Nitin Anand, Michel Cukier,
#William H. Sanders

#University of Maryland and University of Illinois

#URL: http://Ferret.crhc.uiuc.edu

#Permission is hereby granted, free of charge, to any person obtaining a
#copy of this software and associated documentation files (the Software),
#to deal with the Software without restriction, including without
#limitation the rights to use, copy, modify, merge, publish, distribute,
#sublicense, and/or sell copies of the Software, and to permit persons
#to whom the Software is furnished to do so, subject to the following
#conditions:

# * Redistributions of source code must retain the above copyright
#notice, this list of conditions and the following disclaimers.

# * Redistributions in binary form must reproduce the above copyright
#notice, this list of conditions and the following disclaimers in the
#documentation and/or other materials provided with the distribution.

# * Neither the names of University of Maryland or University of
#Illinois, nor the names of its contributors may be used to endorse or
#promote products derived from this Software without specific prior
#written permission.

#THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS
#OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
#MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
#IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR
#ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
#TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
#SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE.
#-----
use English;
use strict;
use Getopt::Long;
use Config;
use User::pwent;
use File::stat;
use User::grent;
use Sys::Hostname;
use lib "lib";
```

```

use Ferretlib;
#-----
sub vprint;
sub printuseage;
#-----
our $verbose = "";
our $plugin_dir;
our $output_plugin_dir;
our $output_plugin;
our $mode;
#-----
my $list;
my $long_desc;
my $priv_gain;
my %keywords;
my @runwords;
my $useos;
my $useos_version;
#-----
# Include the configuration file.
require "Ferret.conf";
#-----
GetOptions("verbose" => \$verbose,
          "help" => sub { printuseage; exit(0) } ,
          "list" => \$list,
          "longdesc" => \$long_desc,
          "privgain" => \$priv_gain,
          "plugindir=s" => \$plugin_dir,
          "output=s" => \$output_plugin,
          "useos=s" => \$useos,
          "useosversion=s" => \$useos_version,
          "mode=s" => \$mode);
#-----
# Enforce running as root, as most plugins will fail if not root.
if($< != 0)
{
    print "ERROR: This program must be run as root (UID 0)\n";
    printuseage;
    exit (-1);
}
#-----
# Keyword to run from command line argument
my @runwords = @ARGV;
my $argc = scalar @ARGV;
#-----
print "Starting Ferret Security Auditing System\n";
#-----
# argc represent Keywords only. They do not represent options like "verbose",
# "help".
if($argc < 1)
{
    print "INFO: Using default \"all\" keywords\n";
    @runwords = ("all");
}
#-----
my $hostname = hostname();

```

```

chomp $hostname;
my $osname = `uname -s`;
chomp $osname;
my $osversion = `uname -r`;
chomp $osversion;
my $date = `date '+%a %-d %b %Y'`;
chomp $date;
my $time = `date '+%-I:%M %p'`;
chomp $time;
#-----
print "I am running on $hostname using $osname version $osversion\n\n";
#-----
if($useos)
{
    print "Setting OS to $useos\n";
    #if user specified OS then it will use that
}
else
{
    $useos = $osname;
    #if user does not specify anything then it takes from command "uname -s"
    #and set the var to that
}
#-----
if ($useos_version)
{
    print "Setting OS version to $useos_version\n";
}
else
{
    $useos_version = $osversion;
}
#-----
vprint "Using plug-in directory \"$plugin_dir\"\n";
#-----
# Get list of plugins from Plugins directory and store in an array to be used
# afterwards
opendir (PLUGS, $plugin_dir) || die "ERROR: Couldn't open plug-in directory \"$plugin_dir\"\n";
my @plugin_list = grep m/.*\plugin$/, readdir(PLUGS);
closedir PLUGS;
#-----
# Find list of users and groups registered on the host
my @users_array;
my @groups_array;
my @user_name_array;
my @group_id_etc_passwd;
my @group_name_array;
my @group_id_etc_group;
# declaring counter variables ($i, $j, $k, $l)
my $i = 0;
my $j = 0;
my $k;
my $l;
my @var;
my @user_array;
my @group_array;

```



```

my $name;

while(($name) = getpwent)
{
    @users_array = @{$name};
    $user_name_array[$i] = $users_array[0];
    $group_id_etc_passwd[$i] = $users_array[3];
    $i++;
}

while(($name) = getgrent)
{
    @groups_array = @{$name};
    $group_name_array[$j] = $groups_array[0];
    $group_id_etc_group[$j] = $groups_array[2];
    $j++;
}
endpwent();
#-----
# Build a hash with information about the available plug-ins for easy
# reference later.
my @words;
my $word;
my $line;
my %plugins;
my $plugin;
my $def_version;
my $def_keywords;
my $def_oslist;
my $def_short_desc;
my $def_long_desc;
my $def_priv_gain;
my $max_plugin_list;
my $i; #variable used in "for" loop
$max_plugin_list = $#plugin_list;
#-----
for($i=0;$i<=$max_plugin_list;$i++)
{
    $def_version = 0;
    $def_keywords = 0;
    $def_oslist = 0;
    $def_short_desc = 0;
    $def_long_desc = 0;
    $def_priv_gain = 0;
    $plugin= $plugin_list[$i];
    $plugin =~ s/(.*)\.plugin$/\1/;
#-----
    open FILE, "$plugin_dir/$plugin.plugin --info 2>&1 |" || die "ERROR: Couldn't run plug-in
\"$plugin\"\n";

    INFOLOOP: while($line = <FILE>)
    {
        @words = split (/s+/, $line);
        $word = shift (@words);
#-----
        if($word =~ m/VERSION/)

```

```

{
    $plugins{$plugin}{VERSION} = shift (@words);
    vprint "Version number of $plugin: $plugins{$plugin}{VERSION}\n";
    $def_version = 1;
    next INFOLOOP;
}
#-----
if($word =~ m/KEYWORDS/)
{
    $plugins{$plugin}{KEYWORDS} = join (' ', @words);
    vprint "Keywords for $plugin: $plugins{$plugin}{KEYWORDS}\n";

    # Build a hash with keywords from the plugins for keys and
    # the names of the plugins as values, only if this plug-in
    # is for this operating system

    while ($word = shift(@words))
    {
        push (@{$keywords{$word}}, $plugin);
    }
    push (@{$keywords{"all"}}, $plugin);
    $def_keywords = 1;
    next INFOLOOP;
}
#-----
if($word =~ m/OSLIST/)
{
    $plugins{$plugin}{OSLIST} = join (' ', @words);
    vprint "Operating system list for $plugin: $plugins{$plugin}{OSLIST}\n";
    $def_oslist = 1;

    if($plugins{$plugin}{OSLIST} =~ m/$useos/)
    {
        $plugins{$plugin}{ACTIVE} = 1;
    }
    next INFOLOOP;
}
#-----
if($word =~ m/SHORTDESC/)
{
    $plugins{$plugin}{SHORTDESC} = join (' ', @words);
    vprint "Short description of $plugin: $plugins{$plugin}{SHORTDESC}\n";
    $def_short_desc = 1;
    next INFOLOOP;
}
#-----
if($word =~ m/LONGDESC/)
{
    $plugins{$plugin}{LONGDESC} = join (' ', @words);
    vprint "Long description of $plugin: $plugins{$plugin}{LONGDESC}\n";
    $def_long_desc = 1;
    next INFOLOOP;
}
#-----
if($word =~ m/PRIVGAIN/)

```

```

    {
        $plugins{$plugin}{PRIVGAIN} = join ( ' ', @words);
        vprint "Privilege gain of $plugin: $plugins{$plugin}{PRIVGAIN}\n";
        $def_priv_gain = 1;
        next INFOLOOP;
    }
#-----
if($word =~ m/^M/)
{
    print "hello i m caught here \n";
}

}
#-----
if(!$def_version)
{
    print "WARNING: No version defined for plug-in $plugin\n";
    print "    Using default value of 1.0\n";
    $plugins{$plugin}{VERSION} = "1.0";
}

if(!$def_keywords)
{
    print "WARNING: No keywords defined for plug-in $plugin\n";
    print "    Using default value of \"all\"\n";
    # Don't actually need to assign it to all, as it is by default
}

if(!$def_oslist)
{
    print "WARNING: No operating system list for plug-in $plugin\n";
    print "    Using default value of \"Linux\"\n";
    $plugins{$plugin}{OSLIST} = "Linux";
}

if(!$def_short_desc)
{
    print "WARNING: No short description supplied for plug-in $plugin\n";
    print "    Using default value of \"No description\"\n";
    $plugins{$plugin}{SHORTDESC} = "No description";
}

if(!$def_long_desc)
{
    print "WARNING: No long description supplied for plug-in $plugin\n";
    print "    Using default value of \"No description\"\n";
    $plugins{$plugin}{LONGDESC} = "No description";
}

if(!$def_priv_gain)
{
    print "WARNING: No privilege gain supplied for plug-in $plugin\n";
    print "    Using default value of \"No description\"\n";
    $plugins{$plugin}{PRIVGAIN} = "No description";
}

```

```

}
#-----
foreach $plugin(@plugin_list)
{
    $plugin =~ s/(.*)\.plugin$/\1/;
}
#-----
# Remember @runwords? We got it from the command line argument.
# Use that to get the list of plugins we'll run out of the keywords hash
# we just built
#-----
vprint "Keywords are \"" . join (' ', @runwords) . "\"\n";
my $runword;
my @runlist;
#-----
foreach $runword (@runwords)
{
    if(!$keywords{$runword})
    {
        print "WARNING: No plug-ins match keyword \"$runword\"\n";
    }
    else
    {
        # Get a list of the plugins that match the keywords
        push (@runlist, @{$keywords{$runword}});
    }
}

@runlist = sort @runlist;
my @runlist_active;
foreach $plugin(@runlist)
{
    $plugin =~ s/(.*)\.plugin$/\1/;
    if($plugins{$plugin}{ACTIVE} == 1)
    {
        push(@runlist_active,$plugin);
    }
}
#-----
# If we just wanted to list the plug-ins, do that (with descriptions)
# and exit. If keyword(s) are supplied, only print plug-ins that match
# those keywords.
if($list)
{
    if(scalar @runlist_active >= 1)
    {
        print "Plugins available:\n";
        foreach $plugin (@runlist_active)
        {
            $plugin =~ s/(.*)\.plugin$/\1/;
            print "$plugin\n";
            print " $plugins{$plugin}{SHORTDESC}\n";
        }
    }
    exit(0);
}

```

```

#-----
if($long_desc)
{
  if(scalar @runlist_active >= 1)
  {
    print "Plugins available:\n";
    foreach $plugin (@runlist_active)
    {
      $plugin =~ s/(.*)\,plugin$/\1/;
      print "$plugin\n";
      print " $plugins{$plugin}{LONGDESC}\n";
    }
  }
  exit(0);
}
#-----
if($priv_gain)
{
  if(scalar @runlist_active >= 1)
  {
    print "Plugins available:\n";
    foreach $plugin (@runlist_active)
    {
      $plugin =~ s/(.*)\,plugin$/\1/;
      print "$plugin\n";
      print " $plugins{$plugin}{SHORTDESC}\n";
      print " $plugins{$plugin}{PRIVGAIN}\n";
    }
  }
  exit(0);
}
#-----
vprint "Plug-in list to run: @runlist_active\n";
# Run the selected plugins and save the output in another hash
my %result;
my $num_vuln = 0;
my $endtime = 0;
my $elapsed = 0;
my $totaltime = 0;
my $j=0;
my $max_runlist;
$max_runlist = $#runlist_active;
our @vul_plugins;
our @non_vul_plugins;
#-----
print "HOSTNAME $hostname\n";
print "USEOS $useos\n";
print "USEOSVERSION $useos_version\n";
print "DATE $date\n";
print "TIME $time\n\n";
#-----
if($mode =~ m/pvg/)
{
  print "Listing Users and Groups\n";
  for($k = 0; $k <= $#group_id_etc_passwd; $k++)
  {

```

```

for($l = 0;$l <= $#group_id_etc_group;$l++)
{
    if($group_id_etc_passwd[$k] == $group_id_etc_group[$l])
    {
        print "USER $user_name_array[$k] , GROUP $group_name_array[$l] \n";
    }
}
}
print "\n\n";
}

```

```

# Run the Plugins and collect the information from the plugins "output" array
for($j=0; $j <= $max_runlist; $j++)
{
    # All the plugins will be run from here. The runlist was obtained by either
    # using keywords from the user or by default. All the plugins will be run.
    $plugin = $runlist_active[$j];
    vprint "Running $plugin_dir/$plugin\n";
    $plugins{$plugin}{TIMESTAMP} = time;
#-----
    open RUN, "$plugin_dir/$plugin.plugin 2>&1" || die "Error could nt open for run
$plugin_dir/$plugin.plugin\n";
    @{$plugins{$plugin}{OUTPUT}} = <RUN>;
    chomp @{$plugins{$plugin}{OUTPUT}};
    close RUN;
#-----
    $sendtime = time;
    $elapsed = ($sendtime - $plugins{$plugin}{TIMESTAMP});
    $totaltime = $totaltime + $elapsed;
#-----
    vprint "Elapsed time: $elapsed seconds\n";
#-----
}
#-----
# From the output obtained by the plugins divide the plugins into two
# families: Plugins having found vulnerabilities and those plugins which
# could not found any vulnerability
for($j=0; $j <= $max_runlist; $j++)
{
    $plugin = $runlist_active[$j];
    my $line;

    foreach $line(@{$plugins{$plugin}{OUTPUT}})
    {
        if($line =~ m/RESULT/)
        {
            $line =~ s/RESULT//;
            $result{$plugin}{RESULT} = $line;
            chomp($result{$plugin}{RESULT});
            if(($result{$plugin}{RESULT}) =~ m/No Vulnerability Found$/)
            {
                push(@non_vul_plugins,$plugin);
            }
        }
        else
        {
            push(@vul_plugins,$plugin);
        }
    }
}

```

```

        $num_vuln = $num_vuln +1;
    }
}
}
}
#-----
# The plugins having found vulnerabilities are calling the output plugin from
# here in the mode chosen by the user
print "Plugins finding Vulnerabilities are:\n";

my $max_vul_plugins;
$max_vul_plugins = $#vul_plugins;
vprint "max_vul_plugins: $max_vul_plugins\n";
my $counter_var = 0;

for($counter_var=0;$counter_var<=$max_vul_plugins;$counter_var++)
{
    my $vul_plugin;
    $vul_plugin = $vul_plugins[$counter_var];
#-----
# Plugins having found the vulnerabilities are now calling the output plugin
# in the first mode
if($mode =~ /pvg/)
{
    open OUTPUT, "|$output_plugin_dir/$output_plugin" || die "could not open up the output plugin
'$output_plugin_dir/$output_plugin' \n";
    print OUTPUT "LIST PLUGINNAME $vul_plugin\n";
    print OUTPUT "DESCRIPTION $plugins{$vul_plugin}{SHORTDESC}\n";
    print OUTPUT "RESULT $result{$vul_plugin}{RESULT}\n";
#-----
# The output from the individual plugins are printed from here.
my $line;
my @output_array = "";

foreach $line (@{$plugins{$vul_plugin}{OUTPUT}})
{
    chomp($line);
    if($line =~ m/^(VUL|NOT_EXIST)/)
    {
        $line =~ s/VUL//g;
        $line =~ s/NOT_EXIST//g;
        push(@output_array, "$line:");
    }
}

my $max_output_array = $#output_array;
$output_array[$max_output_array] =~ s://;
print OUTPUT "OUTPUT @output_array\n";
print OUTPUT "\nEND\n";
close OUTPUT;
}
#-----
# Plugins having found the vulnerabilities are now calling the output plugin
# in the second mode
if($mode =~ /pvg/)
{

```

```

    open OUTPUT, "|$output_plugin_dir/$output_plugin" || die "could not open up the output plugin
'$output_plugin_dir/$output_plugin'\n";
#-----
    # The output from the individual plugins are printed from here.
    my @pvginfo;
    my $line;
    foreach $line (@{$plugins{$vul_plugin}{OUTPUT}})
    {
        chomp($line);
        if($line =~ m/^NOT_EXIST/)
        {
            $line =~ s/NOT_EXIST//g;
            print OUTPUT "NOFILE $vul_plugin, $line\n";
        }
    }
    foreach $line (@{$plugins{$vul_plugin}{OUTPUT}})
    {
        chomp($line);
        if($line =~ m/^PRIV/)
        {
            $line =~ s/PRIV//g;
            print OUTPUT "PRIVGAIN $vul_plugin, $line\n\n";
        }
    }
    close OUTPUT;
}
}
#-----
# Those plugins which could not find any vulnerability are calling the
# output plugin from here in the mode chosen by the user

print "\nPlugins not finding any vulnerability are:\n";

my $max_non_vul_plugins=0;
$max_non_vul_plugins = $#non_vul_plugins;
vprint "max_nonvul_plugins: $max_non_vul_plugins\n";

for($counter_var=0;$counter_var<= $max_non_vul_plugins;$counter_var++)
{
    my $non_vul_plugin;
    $non_vul_plugin = $non_vul_plugins[$counter_var];
#-----
    # Plugins not having found vulnerabilites are now calling the output plugin
    # in the first mode
    if($mode !~ /pvg/)
    {
        open OUTPUT, "|$output_plugin_dir/$output_plugin" || die "could not open up the output plugin
'$output_plugin_dir/$output_plugin'\n";
        print OUTPUT "NON_VUL PLUGINNAME $non_vul_plugin\n";
        print OUTPUT "DESCRIPTION $plugins{$non_vul_plugin}{SHORTDESC}\n";
        print OUTPUT "RESULT $result{$non_vul_plugin}{RESULT}\n";
#-----
        # The output from the individual plugins are printed from here.
        my $line;
        my @output_array = "";
        foreach $line (@{$plugins{$non_vul_plugin}{OUTPUT}})

```



```

{
  chomp($line);
  if($line =~ m/^(VUL|NOT_EXIST)/)
  {
    $line =~ s/VUL//g;
    $line =~ s/NOT_EXIST//g;
    push(@output_array, "$line:");
  }
}
my $max_output_array = $#output_array;
$output_array[$max_output_array] =~ s://;
print OUTPUT "OUTPUT @output_array\n";
print OUTPUT "END\n";
close OUTPUT;
}
}
#-----
# Plugins not having found vulnerabilities are now calling the output plugin
# in the second mode
if($mode =~ /pvg/)
{
  open OUTPUT, "|$output_plugin_dir/$output_plugin" || die "could not open up the output plugin
'$output_plugin_dir/$output_plugin' \n";
  print OUTPUT "NON_VUL $non_vul_plugin\n";
}
close OUTPUT;
}

print "$num_vuln vulnerabilities found in $totaltime seconds\n";

#####
# End of main program
#####

# Print out the usage information. Note that this does not
# end the program, should you want to do something AFTER
# printing the usage.
sub printusage
{
  print "Usage: Ferret [options] [keyword [keyword ...]]\n";
  print "--mode      : \t This argument can be set to 'vul' or 'pvg'.
                By default, the argument is set to 'vul'.
                With 'vul' Ferret lists all the
                vulnerabilities found. With 'pvg' Ferret
                lists all the vulnerabilities found and the
                associated privilege gains.\n";
  print "--list      : \t This argument lists all the vulnerability
                checking plugins with their short
                descriptions.\n";
  print "--longdesc   : \t This argument lists all the vulnerability
                checking plugins with their long descriptions.\n";
  print "--privgain   : \t This argument lists all the vulnerability
                checking plugins with the possible privilege
                changes if the vulnerabilities found by those
                plugins are exploited.\n";
  print "--output    : \t This argument sets the output plugin to be
                used. By default the argument is set to

```

```

        'outputplugin_common_ascii'.\n";
print "--help      :\t This arguement lists the help options.\n";
print "--useos     :\t This arguement sets the system OS name to
        that specified by the user.\n";
print "--useosversion :\t This arguement sets the system OS version
        to that specified by the user.\n";
print "keywordlist :\t This arguement specify keywords to
        run Ferret.\n";
print "           \t Note that the keywords must be after all
        optional arguments.\n";
}

# Print out additional information, if verbose mode is turned on but
#take care to see in which mode Ferret is running and so display relevant
#information to that type only

sub vprint
{
    if($verbose)
    {
        print @_;
    }
}

```

Appendix A.2: Source code of Vulnerability Checking Plug-ins

The source code for some of the vulnerability checking plug-ins from different plug-in families are given below

1. bin-root-ownership.plugin

```
eval '(exit $?0)' && eval 'exec perl -S $0 ${1+"$@"}'  
  & eval 'exec perl -S $0 $argv:q'  
  if 0;
```

```
#Copyright 2002, 2003, 2004, University of Maryland and University of  
#Illinois. All rights reserved.
```

```
#Developed by: Jason Martin, Anil Sharma, Nitin Anand, Michel Cukier,  
#William H. Sanders
```

```
#University of Maryland and University of Illinois
```

```
#URL: http://Ferret.crhc.uiuc.edu
```

```
#Permission is hereby granted, free of charge, to any person obtaining a  
#copy of this software and associated documentation files (the Software),  
#to deal with the Software without restriction, including without  
#limitation the rights to use, copy, modify, merge, publish, distribute,  
#sublicense, and/or sell copies of the Software, and to permit persons  
#to whom the Software is furnished to do so, subject to the following  
#conditions:
```

```
# * Redistributions of source code must retain the above copyright  
#notice, this list of conditions and the following disclaimers.
```

```
# * Redistributions in binary form must reproduce the above copyright  
#notice, this list of conditions and the following disclaimers in the  
#documentation and/or other materials provided with the distribution.
```

```
# * Neither the names of University of Maryland or University of  
#Illinois, nor the names of its contributors may be used to endorse or  
#promote products derived from this Software without specific prior  
#written permission.
```

```
#THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS  
#OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
#MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  
#IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR  
#ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,  
#TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE  
#SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE.  
#-----
```

```
use English;  
use strict;  
use Getopt::Long;
```

```

my @plugin_info = (
"VERSION 0.1\n",
"KEYWORDS files directory owner ownership root\n",
"OSLIST Linux SunOS HP-UX\n",
"SHORTDESC Checks whether root owns or not the critical /bin directory \n",
"LONGDESC This plug-in checks whether the critical system directory /bin is owned by root\n",
"PRIVGAIN No privilege change,/bin,not owned by root\n"
);

GetOptions("info" => sub { print @plugin_info; exit(0); });

my @output;
my $user;
my $vulnerability_found = 0;
my $chk_dir;

$chk_dir = '/bin';
if (! -e $chk_dir)
{
    push @output, "NOT_EXIST Directory /bin does not exist\n";
    push @output, "RESULT No Vulnerability Found\n";
    print @output;
    die "Directory /bin does not exist\n";
}

my ($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime, $ctime, $blksize, $blocks) = stat
"$chk_dir";

if( $uid ne 0)
{
    $vulnerability_found = 1;
}

push @output, "VUL None\n";

if ($vulnerability_found == 1)
{
    push @output, "RESULT Vulnerability Found\n";
}
else
{
    push @output, "RESULT No Vulnerability Found\n";
}

push @output, "PRIV No privilege change,/bin,not owned by root\n";
print @output;

```

2. check-access-exports.plugin

```

eval '(exit $?0)' && eval 'exec perl -S $0 ${1+"$@"}'
    & eval 'exec perl -S $0 $argv:q'
    if 0;

```

#Copyright 2002, 2003, 2004, University of Maryland and University of
#Illinois. All rights reserved.

```
#Developed by: Jason Martin, Anil Sharma, Nitin Anand, Michel Cukier,  
#William H. Sanders  
#University of Maryland and University of Illinois  
#URL: http://Ferret.crhc.uiuc.edu  
#Permission is hereby granted, free of charge, to any person obtaining a  
#copy of this software and associated documentation files (the Software),  
#to deal with the Software without restriction, including without  
#limitation the rights to use, copy, modify, merge, publish, distribute,  
#sublicense, and/or sell copies of the Software, and to permit persons  
#to whom the Software is furnished to do so, subject to the following  
#conditions:
```

```
# * Redistributions of source code must retain the above copyright  
#notice, this list of conditions and the following disclaimers.
```

```
# * Redistributions in binary form must reproduce the above copyright  
#notice, this list of conditions and the following disclaimers in the  
#documentation and/or other materials provided with the distribution.
```

```
# * Neither the names of University of Maryland or University of  
#Illinois, nor the names of its contributors may be used to endorse or  
#promote products derived from this Software without specific prior  
#written permission.
```

```
#THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS  
#OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
#MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  
#IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR  
#ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,  
#TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE  
#SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE.  
#-----
```

```
use English;  
use strict;  
use Getopt::Long;
```

```
my @plugin_info = (  
"VERSION 0.1\n",  
"KEYWORDS explicit access control /etc/exports\n",  
"OSLIST Linux \n",  
"SHORTDESC Checks for files in /exports that have been given some access control or not.\n",  
"LONGDESC This plug-in checks for files in /exports have been given some access control or not and",  
" if there is no explicit access control then it shows a warning\n",  
"PRIVGAIN No privilege change, /etc/exports,access not specified\n"  
);
```

```
GetOptions("info" => sub { print @plugin_info; exit(0); } ),
```

```
my @array;  
my $vulnerability_found = 0;  
my $line;  
my $match_line;  
my @output;  
my $chk_file;
```

```

$chk_file = "/etc/exports";

if(! -e $chk_file)
{
    push @output,"NOT_EXIST File $chk_file does not exist\n";
    push @output,"RESULT No Vulnerability Found\n";
    print @output;
    die "File $chk_file does not exist\n";
}

open(EXPORTS, "/etc/exports") || die "could not open file /etc/exports\n";

while (($line=<EXPORTS>) ne "")
{
    $match_line = "";
    $match_line = $line;
    @array = split(/\s+/, $match_line);
    my $max_array = $#array;
    if ($max_array > 1)
    {
        $vulnerability_found = 1;
    }
}

if($vulnerability_found == 1)
{
    push @output,"VUL None\n";
    push @output,"RESULT Vulnerability Found\n";
}
else
{
    push @output,"VUL None\n";
    push @output,"RESULT No Vulnerability Found\n";
}

push @output, "PRIV No privilege change,/etc/exports,access not specified\n";
print @output;

```

3. check-path-cshrc.plugin

```

eval '(exit $?0)' && eval 'exec perl -S $0 ${1+"$@"}'
    & eval 'exec perl -S $0 $argv:q'
    if 0;

```

#Copyright 2002, 2003, 2004, University of Maryland and University of Illinois. All rights reserved.

#Developed by: Jason Martin, Anil Sharma, Nitin Anand, Michel Cukier,
#William H. Sanders

#University of Maryland and University of Illinois

#URL: <http://Ferret.crhc.uiuc.edu>

#Permission is hereby granted, free of charge, to any person obtaining a

```
#copy of this software and associated documentation files (the Software),
#to deal with the Software without restriction, including without
#limitation the rights to use, copy, modify, merge, publish, distribute,
#sublicense, and/or sell copies of the Software, and to permit persons
#to whom the Software is furnished to do so, subject to the following
#conditions:
```

```
# * Redistributions of source code must retain the above copyright
#notice, this list of conditions and the following disclaimers.
```

```
# * Redistributions in binary form must reproduce the above copyright
#notice, this list of conditions and the following disclaimers in the
#documentation and/or other materials provided with the distribution.
```

```
# * Neither the names of University of Maryland or University of
#Illinois, nor the names of its contributors may be used to endorse or
#promote products derived from this Software without specific prior
#written permission.
```

```
#THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS
#OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
#MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
#IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR
#ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
#TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
#SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE.
#-----
```

```
use English;
use strict;
use Getopt::Long;
use lib "lib";
use lib "../lib";
use Ferretlib;
```

```
my @plugin_info = (
"VERSION 0.1\n",
"KEYWORDS paths world writeable writable startup file files\n",
"OSLIST Linux SunOS HP-UX\n",
"SHORTDESC Checks for paths inside /.cshrc for world writability\n",
"LONGDESC This plug-in checks pathnames inside root's startup file /.cshrc for world writability\n",
"PRIVGAIN No privilege change,/.cshrc,path inside /.cshrc world writable\n"
);
```

```
GetOptions("info" => sub { print @plugin_info; exit(0); });
```

```
my $vulnerability_found = 0;
my @output;
my $line;
my @path;
my $path;
my $actual_mode;
my $match_line;
my @vul_paths;
my $path;
```

```

if (! -e ".cshrc")
{
    push @output,"NOT_EXIST File /.cshrc does not exist\n";
    push @output,"RESULT No Vulnerability Found\n";
    print @output;
    die "File /.cshrc does not exist\n";
}

open(CSHRC,".cshrc") || die "could not open file /.cshrc\n";

while(($line=<CSHRC>) ne "")
{
    $match_line = "";
    $match_line = $line;
    if ($line =~ /^set/)
    {
        @path = split(/\s+/, $line);
    }
}

close CSHRC;

substr($path[0],0,12)="";
my $max_path = $#path;
my $i=0;

for($i=4;$i <= $max_path; $i++)
{
    my ($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime,$ctime, $blksize, $blocks) =stat
"/$path[$i]";
    $actual_mode = sprintf "%04o", $mode & 07777;
    if (($mode & 00002))
    {
        push @output, "VUL $path[$i] is world writable (mode $actual_mode):\n";
        push(@vul_paths,"$path[$i]");
        $vulnerability_found =1;
    }
}

if($vulnerability_found == 1)
{
    push @output,"RESULT Vulnerability Found\n";
}
else
{
    push @output,"RESULT None\n";
    push @output,"RESULT No Vulnerability Found\n";
}

foreach $path(@vul_paths)
{
    push (@output, "PRIV No privilege change,/.cshrc,path $path inside /.cshrc world writable\n");
}
print @output;

```


Appendix A.3: Source code of Output Plug-in

The code for the output plug-in is given below

```
eval '(exit $?0)' && eval 'exec perl -S $0 ${1+"$@"}'
  && eval 'exec perl -S $0 $argv:'
  if 0;

#Copyright 2002, 2003, 2004, University of Maryland and University of
#Illinois. All rights reserved.

#Developed by: Jason Martin, Anil Sharma, Nitin Anand, Michel Cukier,
#William H. Sanders

#University of Maryland and University of Illinois

#URL: http://Ferret.crhc.uiuc.edu

#Permission is hereby granted, free of charge, to any person obtaining a
#copy of this software and associated documentation files (the Software),
#to deal with the Software without restriction, including without
#limitation the rights to use, copy, modify, merge, publish, distribute,
#sublicense, and/or sell copies of the Software, and to permit persons
#to whom the Software is furnished to do so, subject to the following
#conditions:

# * Redistributions of source code must retain the above copyright
#notice, this list of conditions and the following disclaimers.

# * Redistributions in binary form must reproduce the above copyright
#notice, this list of conditions and the following disclaimers in the
#documentation and/or other materials provided with the distribution.

# * Neither the names of University of Maryland or University of
#Illinois, nor the names of its contributors may be used to endorse or
#promote products derived from this Software without specific prior
#written permission.

#THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS
#OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
#MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
#IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR
#ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
#TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
#SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE.
#-----

use English;
use strict;
#-----
my $matchline;
require "Ferret.conf";
#-----
while(($matchline = <STDIN>) ne "")
{
```

```
chomp($matchline);

if($matchline =~ m/^LIST/)
{
    $matchline =~ s/LIST //;
    print "$matchline\n";
}

if($matchline =~ m/^RESULT/)
{
    print "$matchline\n";
}

if($matchline =~ m/^DESCRIPTION/)
{
    print "$matchline\n";
}

if($matchline =~ m/^OUTPUT/)
{
    print "$matchline\n";
}

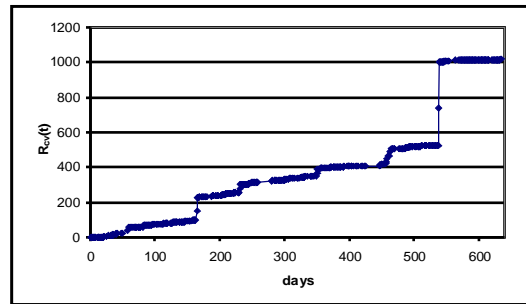
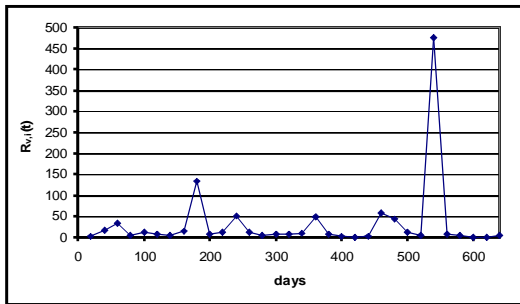
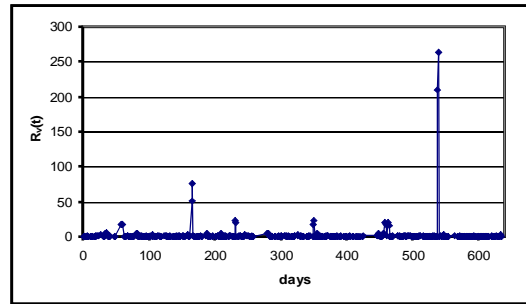
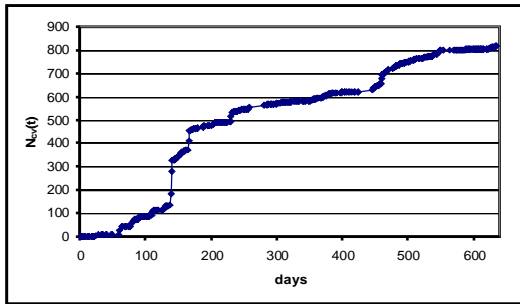
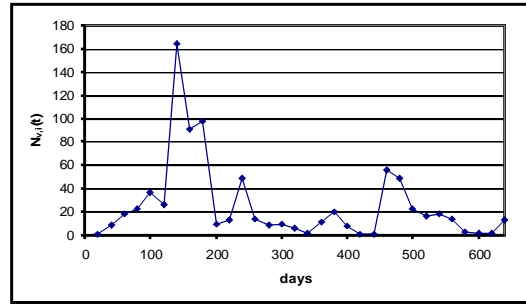
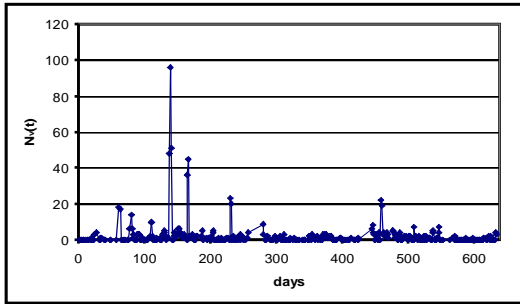
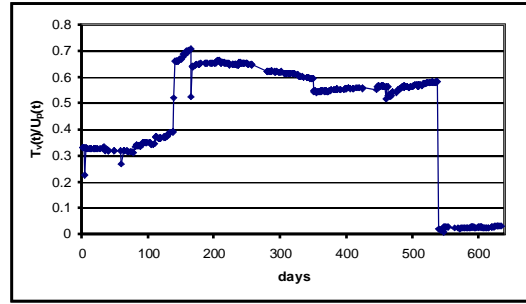
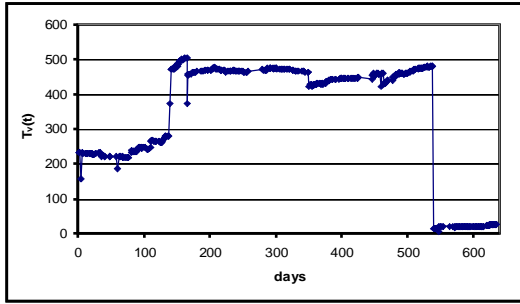
if($matchline =~ m/^NOFILE/)
{
    $matchline =~ s/NOFILE//;
    print "$matchline\n";
}

if($matchline =~ m/^PRIVGAIN/)
{
    $matchline =~ s/PRIVGAIN//;
    print "$matchline\n";
}

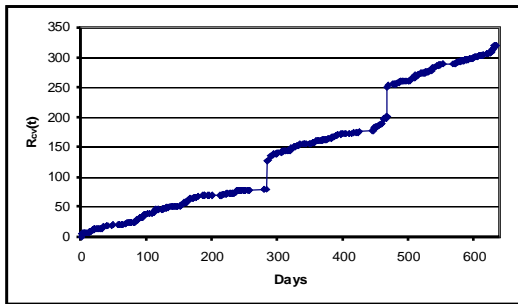
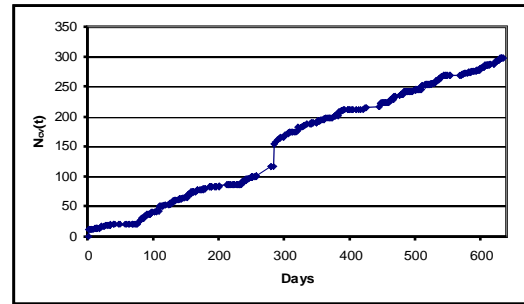
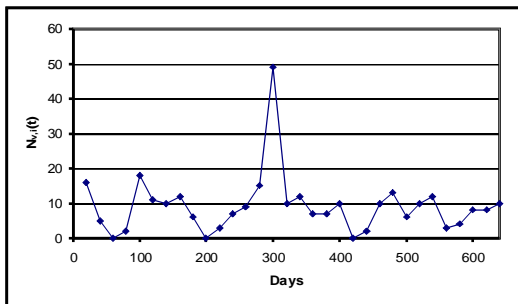
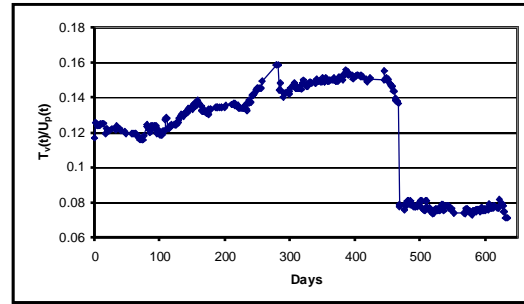
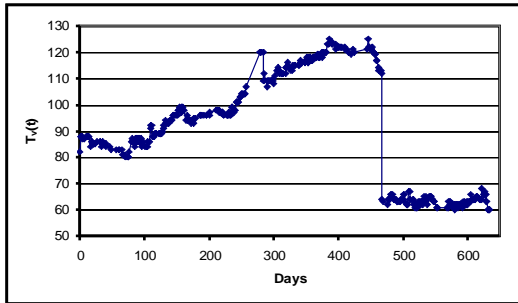
if($matchline =~ m/^NON_VUL/)
{
    $matchline =~ s/NON_VUL //;
    print "$matchline\n";
}

if($matchline =~ m/^END/)
{
    $matchline =~ s/END//;
    print "$matchline\n";
}
}
```

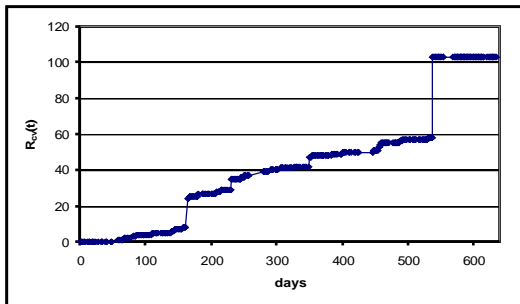
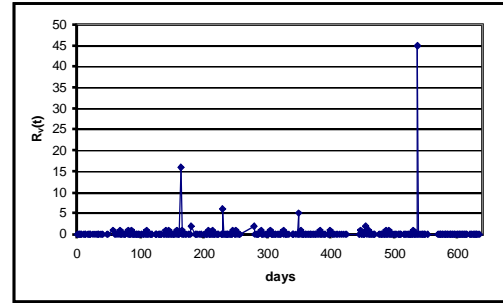
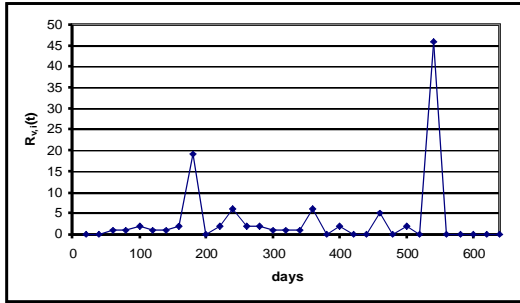
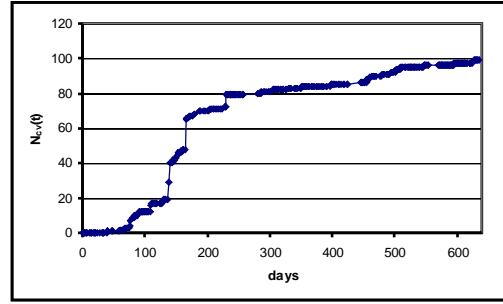
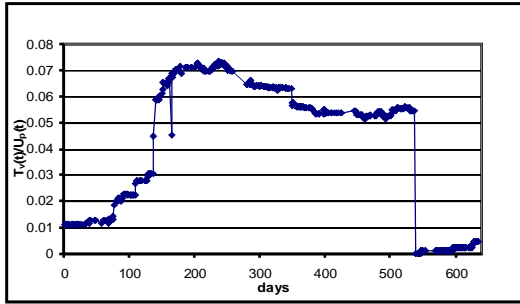
Appendix B.1: Plots Obtained for Group A



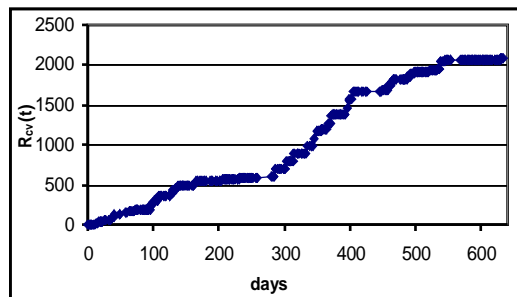
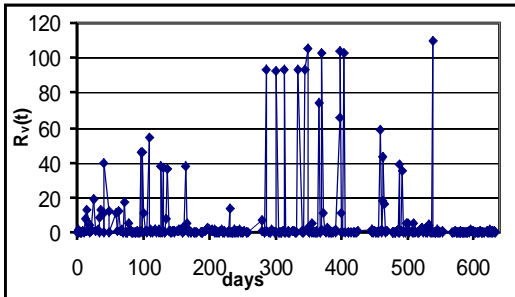
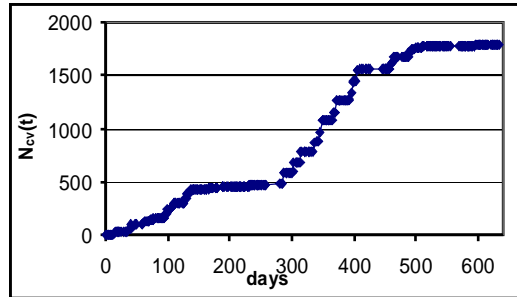
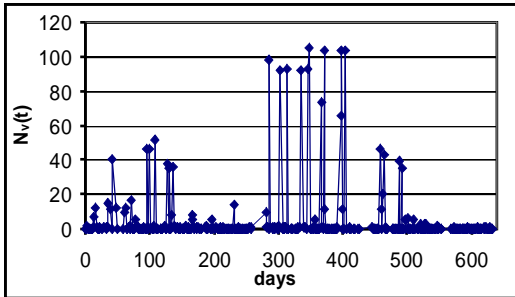
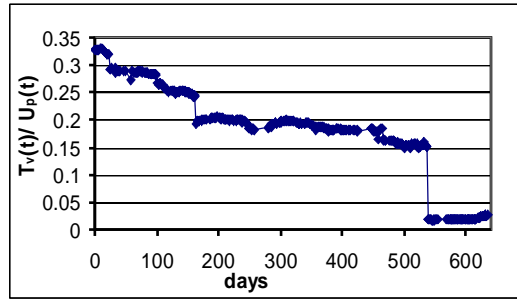
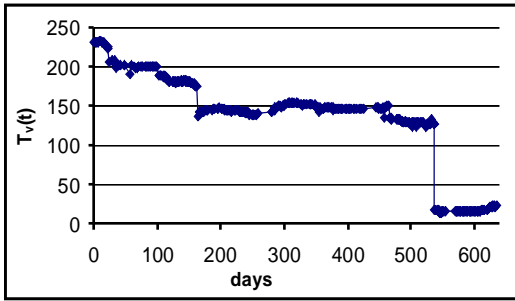
Appendix B.2: Plots Obtained for Group B



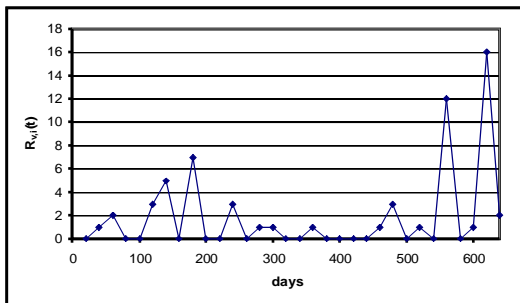
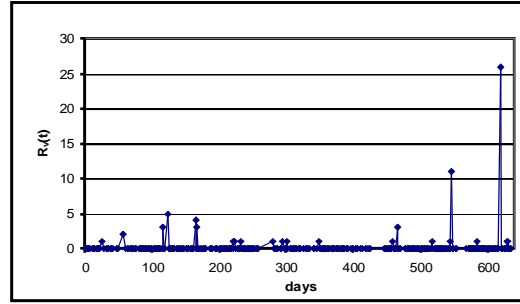
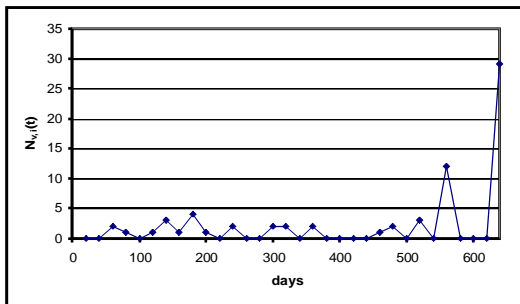
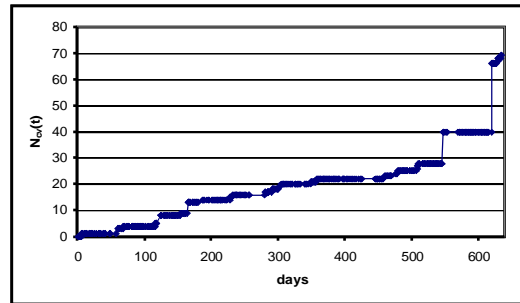
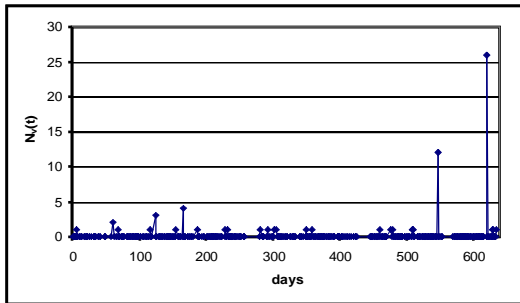
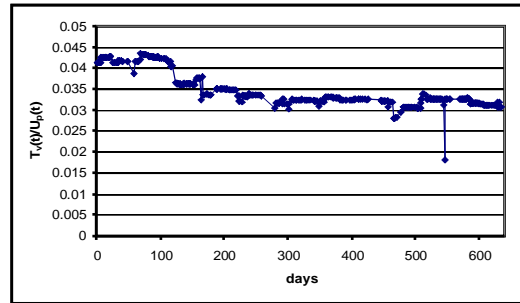
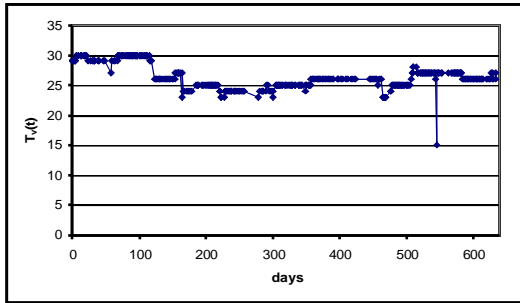
Appendix B.3: Plots Obtained for Group E



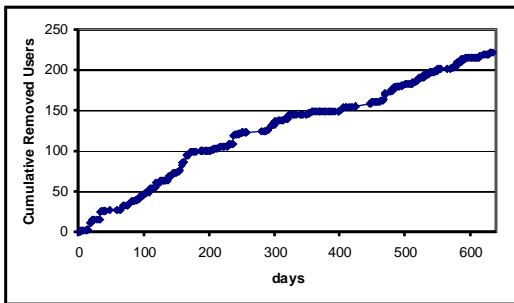
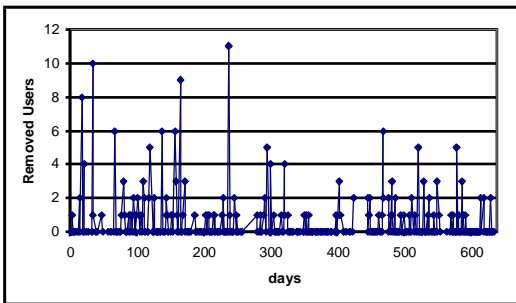
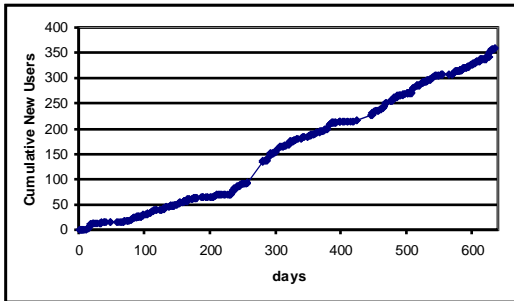
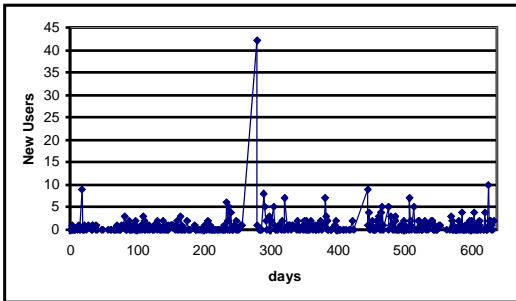
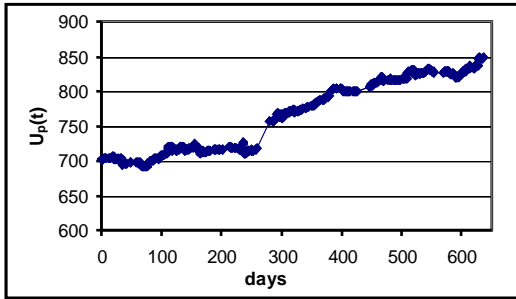
Appendix B.4: Plots Obtained for Group G



Appendix B.5: Plots Obtained for Group H



Appendix B.6: Users Related Charts



Appendix C.1: Perl Script To Transfer Output Data to Mysql Database

This script will enable data transfer from Ferret text files to Mysql database

```
#!/C:/perl/bin -w

use strict;

use DBI;

my $Ferret_folder;
my @the_files;
my $max;
my $file;
my $i; #counter variable to be used in for loop

    print "Welcome this script will enable You to transfer the data from Ferret text files to Your mysql
    database\n\n";

print "Please specify the folder where Ferret text files are located.\n";

$Ferret_folder = <STDIN>;
chomp($Ferret_folder);

#-----
opendir(INPUT,$Ferret_folder) || die "could n't open up the directory!";
@the_files = readdir(INPUT);
closedir(INPUT);
$max = $#the_files;
#-----

my $date;
my @arr;
my @date;
my @new_date;

#####
# Connect To Database and create the tables for each corresponding output file
#####
my $database;
print "hello please enter the name of the database you want to use\n";
$database = <STDIN>;
chomp($database);

my $hostname = "localhost";
my $username = "xxx";
my $password = "yyy";

    my $dbh = DBI->connect("DBI:mysql:$database:$hostname", $username, $password) or die "Can't
    connect to database: ", $DBI::errstr,"\n";
print "Connected to $database first time... \n";

#tables are automatically named here. the year can be changed
my $year = "Tab_2004_";
```

```

for($i=0; $i <=$max; $i++)
{
    my $sql = $dbh->prepare('create table '.$year.$i.'
(
    id int(11) auto_increment not null,
    PRIMARY KEY (id),
    plugin_name varchar(40) ,
    change_in_privilege varchar(40),
    file_affected varchar(40),
    description varchar(40),
    timestamp varchar(40)
)

');
$sql->execute or die "cant execute SQL statement:", $sql->errstr(), "\n";
}

#get all the tables created in an array to be used later for feeding data
$arr = $dbh->tables();
$dbh->disconnect;

#-----
#####
#Now data will be feeded for each of the files to the tables
#####
for($i=2; $i<= $max;$i++)
{

    my $flag = 0;
    $file = $the_files[$i];
    my @infield = "";
    my $infield;
    my $line = "";

    open(INPUT, "$Ferret_folder/".$file)||die("Can't open report file");
while (<INPUT>)
{
    if(/^From root*/)
    {
        $date = $_;
        @date = split(/\s+/, $date);
        $new_date[0]= $date[4];
        $new_date[1] = $date[3];
        $new_date[2] = $date[6] - 2000;
        $infield = $new_date[0]."-".$new_date[1]."-".$new_date[2];
    }

    if(/^PLUGINS FINDING VULNERABILTIES ARE/)
    {
        $flag = 1;
    }

    if(/^PLUGINS NOT FINDING ANY VULNERABILTIES ARE/)
    {
        $flag = 0;
    }
}

```

```

if($flag == 1)
{
if(!(does not exist) && (!/^PLUGINS FINDING VULNERABILTIES ARE/ )
{
@infield =split(/,/,$_);

#####
# Connect To Database again for feeding the tables
#####
if($#infield >= 3)
{
my $dbh = DBI->connect("DBI:mysql:$database:$hostname", $username, $password);
die "Cannot log into database. Please try again later.\n" unless $dbh;

my $insertid = $dbh->{mysql_insertid};
# Insert values into appropriate table

my $sth = $dbh->prepare("insert into $arr[$i](id, plugin_name, change_in_privilege, file_affected ,
description, timestamp) values ('$insertid', '$infield[0]', '$infield[1]', '$infield[2]', '$infield[3]', '$infield'
)") or die "can't prepare SQL statement:", $dbh->errstr(), "\n";

$sth->execute or die "cant execute SQL statement:", $sth->errstr(), "\n";
$dbh->disconnect;
}
}
}

}#END OF WHILE LOOP

close INPUT;
}#END OF FOR LOOP

exit(0);

```

Appendix C.2: Perl Script to Query Mysql Database

This script queries the Mysql database

```
#!/C:/perl/bin
use DBI;
my $database;
print "hello please enter the name of the Ferret database you want to Query for\n";
$database = <STDIN>;
chomp($database);
#-----
my $hostname = "localhost";
my $username = "anil";
my $password = "";
my $dbh = DBI->connect("DBI:mysql:$database:$hostname", $username, $password) or die
"Can't connect to database: ", $DBI::errstr,"\n";
#-----
#get all the tables of the database in an array to be used later
my @tables = $dbh->tables();
#-----

my @array=("bin-root-ownership","dev-root-ownership","etc-passwd-root-ownership",
"etc-root-ownership","etc-world-writable","group-all-field","group-alphanumeric",
"group-blank-line","group-gid-blank","group-gid-numeric","group-passwd-blank",
"group-passwd-char","homedir-group","homedirs-.bashrc-group-writable",
"homedirs-.bashrc-world-writable","homedirs-.cshrc-group-writable","homedirs-.cshrc-world-
writable");

my $max_array = $#array;

#-----
my $results ='results_2004_';

for($i=0; $i <=$max_array; $i++)
{

    my $sql = $dbh->prepare('create table '.$results.$i.'
        (
            plugin_name varchar(40),
            timestamp varchar(40),
            count int
        )
    ');
    $sql->execute or die "cant execute SQL statement:", $sql->errstr(), "\n";
}

#-----
my @array_tables_three;
my @tables_two = $dbh->tables();

for($p =0; $p <= $max_array; $p++)
{
push(@array_tables_three,$tables_two[$p]);
}
}
```

```
#-----  
  
for(my $j = 0; $j <=$max_array;$j++)  
{  
  for($i = 1; $i <=$#tables; $i++)  
  {  
    my $sth = $dbh->prepare("INSERT INTO $array_tables_three[$j]  
  
SELECT plugin_name, timestamp, count(plugin_name)  
FROM $tables[$i] where plugin_name = '$array[$j]' GROUP BY timestamp") or die "can't  
prepare SQL statement:", $dbh->errstr(), "\n";  
  
$sth->execute or die "cant execute SQL statement:", $sth->errstr(), "\n";  
  
  }  
}  
  
#-----  
$dbh->disconnect;  
  
    exit(0);
```

Appendix C.3: Macro to Plot Graphs in MS-Excel

This Macro plots the graph in MS-Excel

```
Sub Macro_Plot()

Dim co As ChartObject
Dim cw As Long, rh As Long

'Get data for positioning the chart
cw = Columns(1).Width
rh = Rows(1).Height

'Position the chart using column and row height units
Set co = ActiveSheet.ChartObjects.Add(cw * 3, rh * 0.5, cw * 8, rh * 20)

'Name the chart
co.Name = "Chart Example"

'Set chart type
co.Chart.ChartType = xlLine

'Add data series
co.Chart.SeriesCollection.Add _
Source:=ActiveSheet.Range("d2:e67"), _
Rowcol:=xlColumns, SeriesLabels:=True, _
Categorylabels:=True

'Add axes
With co.Chart
    .HasAxis(xlCategory, xlPrimary) = True
    .HasAxis(xlCategory, xlSecondary) = False
    .HasAxis(xlValue, xlPrimary) = True
    .HasAxis(xlValue, xlSecondary) = False
End With

'Axis title formatting
With co.Chart.Axes(xlCategory)
    .HasTitle = True
    .AxisTitle.Caption = "DAYS"
    .AxisTitle.Border.Weight = xlMedium
End With

With co.Chart.Axes(xlValue)
    .HasTitle = True
    With .AxisTitle
        .Caption = "Tv(t) "
        .Font.Size = 10
        .Orientation = xlHorizontal
        .Characters(14, 4).Font.Italic = True
        .Border.Weight = xlMedium
    End With
End With
```

```
co.Chart.Axes(xlCategory).MajorTickMark = xlTickMarkCross
```

```
co.Chart.Axes(xlCategory).TickLabelPosition = _  
xlTickLabelPositionNextToAxis
```

```
With co.Chart.Axes(xlValue)  
.AxisTitle.HorizontalAlignment = xlCenter  
.AxisTitle.VerticalAlignment = xlCenter  
.AxisTitle.ReadingOrder = xlContext  
.AxisTitle.Orientation = xlUpward  
End With
```

```
'Format Chart Title  
co.Chart.HasTitle = True
```

```
With co.Chart.ChartTitle
```

```
.Caption = "home-directories related vulnerabilities on host A "  
.Font.Size = "7"  
.Characters.Font.FontStyle = "Bold"  
End With  
.....
```

```
co.Chart.PlotArea.Interior.ColorIndex = 0
```

```
End Sub
```

References

- [1] C. Cachin, J. Camenisch, M. Dacier, Y. Deswarte, J. Dobson, D. Horne, K. Kursawe, J. C. Laprie, J. C. Lebraud, D. Long, T. McCutcheon, J. Muller, et al., MAFTIA Reference Model and Use Cases, MAFTIA deliverable D2, 2000.
- [2] CERT, http://www.cert.org/stats/cert_stats.html#vulnerabilities
- [3] COPS, <http://www.fish.com/cops>
- [4] M. Dacier and Y. Deswarte, Privilege Graph: an Extension to the Typed Access Matrix Model, in Proc. Third European Symposium Research in Computer Security (ESORICS94), pp. 317-334, 1994.
- [5] M. Dacier and Y. Deswarte, and M. Kaaniche, Quantitative Assessment of Operational Security: Models and Tools, LAAS Research Report 96493, May 1996.
- [6] D. Farmer and E. H. Spafford, The COPS Security Checker System, in Proc. Summer Usenix Conference, Berkeley, CA, USA, pp. 165-170, 1990.
- [7] Ferret, <http://ferret.crhc.uiuc.edu>
- [8] S. Garfinkel and G. Spafford, Practical UNIX and Internet Security, Second Edition, April 1996
- [9] D. Gollmann , Computer Security, John Wiley & Sons, 2001.
- [10] John the Ripper password cracker, <http://www.openwall.com/john/>
- [11] A. Muffett, Crack, 1992. Available via anonymous ftp from cert.org
- [12] Nessus, <http://www.nessus.org>
- [13] R. Ortalo, Y. Deswarte, and M. Kaaniche, Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security, in IEEE Transactions on Software Engineering, vol. 25, no. 5, pp. 633-650, Sept.-Oct. 1999.
- [14] Passwords, <http://www.free-definition.com/Password.html>
- [15] A. Sharma, J. R. Martin, N. Anand, M. Cukier, and W. H. Sanders, Ferret: A Host Vulnerability Checking Tool, Pacific Rim Dependable Computing, 2004.
- [16] Tiger Analytical Research Assistant, <http://www.arc.com.tara/index.shtml>.
- [17] H. Tsitsivas, UNIX System Management and Security: Differences between Linux, Solaris, AIX and HP-UX, GSEC Practical Assignment, Version 1.4b, www.giac.org/practical/GSEC/Haral_Tsitsivas_GSEC.pdf
- [18] UNIX, <http://www.unix-systems.org>
- [19] R. Zadajmool, The Science of Host Based Security, http://www.windowsecurity.com/articles/Science_Host_Based_Security.html