# Teaching Hands-On Linux Host Computer Security

ROSE SHUMBA
Indiana University of Pennsylvania (IUP)

In the summer of 2003, a project to augment and improve the teaching of information assurance courses was started at IUP. Thus far, ten hands-on exercises have been developed. The exercises described in this article, and presented in the appendix, are based on actions required to secure a Linux host. Publicly available resources were used to develop the exercises, which have been successfully utilized since spring 2003 to teach cybersecurity basics classes. The experiences and challenges encountered in teaching the course and possible future work are also described.

Categories and Subject Descriptors: K.3.2 **[Computers and Education]**: Computer and Information Science Education – *Curriculum*

General Terms: Security

Additional Key Words and Phrases: Computer security, host security exercises, user accounts, file permissions, cryptography, file integrity

## 1. INTRODUCTION

Given the complexity of current attacks, the shortage of information assurance professionals, and rapid rate of change in high-technology fields, there is need to develop more effective methods of teaching information assurance courses.

During the summer of 2003, with funding from IUP, a project was started to augment and improve the teaching of a cybersecurity basics course, which is an interdisciplinary course for criminology, business, and communications media majors. The project involved the evaluation of the effectiveness of available host security tools, including those previously used in teaching the course, development of hands-on exercises based on the tools evaluated, and the integration of the hands-on exercises and the theories and principles of the cybersecurity basics course.

The course is based on techniques for hardening a Linux host server. Linux was selected because the source code is freely available and there is a good base of freely available security tools and utilities. The main textbook for the course was Garfinkel et al.'s [2003], *Practical UNIX & Internet Security*. An isolated network security laboratory, with 22 Red Hat Linux 9.2 servers, 10 Cisco 2600 series routers, and 10 PIX firewalls, was dedicated for the purposes of teaching the course.

At the beginning of the semester, each student is assigned a Linux server for which he/she becomes a system security administrator. The student is responsible for maintenance, including the hardening of the assigned server.

In this article, we first introduce the background of the course and related work. For each exercise, we describe the pedagogical and practical objectives, the background of

This research was supported by Indiana University of Pennsylvania.
Contact author's address: Rose Shumba, Computer Science Department, Indiana University of PA, Indiana, PA 15701

the students, the required software, and provide a qualitative discussion. Section 4 describes the experiences and challenges encountered in teaching the course, and shortcomings of, this project. The article concludes with a description of possible future work.

## 2. BACKGROUND OF THE COURSE AND RELATED WORK

Many courses in computer security focus on what computer scientists need to know to develop security algorithms and products. However, computer science graduates will not go out and invent new encryption algorithms or fire filtering processes; but they will need to know how to

- purchase the right security equipment, install it, configure it, and keep it working;
- create security plans, policies, and architectures that will reflect their specific companies' needs and leave them without weak links in their security; and
- manage security on a daily basis and know what to do when security breaches occur.

The main objective of the cybersecurity course is to introduce students to the basic elements necessary to defend a Linux host. Students analyze the security building blocks for a Linux host and how to contribute to a high level of host security. The security blocks covered in the theoretical part of the course include the following:

- *Users, Passwords, and Authentication* discusses the purpose of passwords, explains what elements make good or bad passwords, and describes how the crypt() password encryption system works. It also introduces the role of the superuser and the pluggable authentication module system.
- *Users, Groups, and Superusers* explains how Linux groups can be used to control access to files and devices.
- *Filesystems and Security* covers the security provisions of the Linux file system and explores how to restrict access to files and directories to the file owner, to a group of people, or to all users of the computer system.
- *Cryptography Basics* discusses the roles of encryption and message digests in security.
- *Defending Accounts* describes ways in which an attacker might break into a computer system.
- *Integrity Management* discusses how to monitor the file system for unauthorized changes; the configuration and use of Tripwire is also covered.
- *Auditing and Logging* explains the logging mechanism that Linux provides to help audit the usage and behavior of the system. and
- *Vulnerability Assessment* describes several tools for assessing vulnerabilities .

Protecting the servers is ***critical.*** The security blocks, explained above, form a sound defense strategy made up of multiple layers. If one layer is penetrated, there are other protective layers. The material covered in this course is useful to both users and system security administrators.

The hands-on exercises complement the security block concepts. To teach computer security adequately requires hands-on exercises so that the students can experiment with new and interesting tools. The labs require an isolated lab running at least Red Linux 8.2; the tools required are all publicly available.

On completion of this course, students will

- know how to defend a given Linux host;
- be aware of the potential computer security threats and alternative methods of prevention and detection;
- be aware of the "best of breed" tools and "best practices" in host security; and
- understand the basic operations of a Linux machine.

Although Microsoft Windows is the most-used operating system, we decided to use Linux because (1) Linux is free; (2) there is an abundance of freely downloadable security tools (we used free downloads from two main sites: www.cert.org and www.insecure.org); and (3) tutorials and documentation are available with most of the tools, which can be tailored to individual teaching environments.

The course does not cover physical security. The prerequisite for the course is the COSC110 (problem solving and structured programming) or the instructor's approval. The cybersecurity basics course is a prerequisite for the network security course.

There have been other efforts to develop lab exercises. Most were based on an attack-defense strategy by means of penetration test competitions. Penetration test competitions are not a new idea; they have been used often in the computer security curriculum as a way for students to demonstrate cumulative understanding of the various security issues, strategies, and tools discussed during the semester [Wagner and Wudi 2004]. One of the best-known competitions is the annual cyberwar competition between the four branches of the United States military academies [Jackson 2002]. Other cyberwar laboratory structures, courses, and exercises have been presented at SIGSCE conferences [Mateti 2003; Micco and Rossman 2002]. Although these competitions provide students with an opportunity to experimentally design realistic attack-defense mechanisms and draw conclusions based on their exploration, it is the author's opinion that students should only be taught how to defend and not how to attack.

## 3. THE HANDS-ON EXERCISES

### Exercise One: Introductory Exercise

*Pedagogical objective*: To understand the basics of the Linux operating system.

*Practical objective*: To familiarize students with the basic activities on the Linux machine (i.e., gaining root privileges, mounting devices, and some of the commonly used Linux commands).

*Student background*: This exercise is designed specifically for noncomputer science majors who have no previous Linux experience.

*Qualitative discussion*: Students must master the activities in this exercise to be able to complete the rest of the exercises. Many computer science and Management Information Systems (MIS) majors quickly become bored with this exercise, but it presents a challenge to the rest of the students, since they take much longer to complete it.

### Exercise Two: Users, Passwords, and Authentication I

*Pedagogical objective*: To defend a given Linux host.

*Practical objective*: To introduce students to good security for an account.

*Student background*: Students should have knowledge of user, password, and authentication concepts; good practice in creating user accounts; how to get root privileges; and how password-cracking programs work.

*Required facilities and software*: John the Ripper[1] and Nutcracker.[2]

*Explanation*: The first line of defense against system abuse is good account security. People trying to gain unauthorized access to any system often try to acquire the usernames and passwords of legitimate users while on their way to exploiting the root account. Lax local security can allow them to "upgrade" their normal user access to root access using a variety of bugs and poorly set up local services. On completion of this exercise, students will know how to

- take good care of their own account by changing passwords regularly (including the root passwords);
- follow good practices in the creation of user accounts; there are quite a number of web sites with guidelines for choosing passwords:
  - http://www.alw.nih.gov/Security/Docs/passwd.html
  - http://www.id-telephony.com/docs/choosing-good passwords.html
  - http://www.equity.psu.edu/nis/pwdsel.html
- audit existing passwords by running cracking programs (John the Ripper [3] and Nutcracker [4]).

*Qualitative discussion*: Students create their own accounts and test them by running a password cracker. There is much excitement as students crack their account passwords. The Nutcracker program takes a shorter time to run than does John the Ripper. Students are keen to know why the two run so differently. Students appreciate the importance of running a password cracker.

### Exercise Three: Users, Passwords, and Authentication II

*Pedagogical objective*: To defend a given Linux host by means of users, passwords, and authentication.

*Practical objective*: To expose students to other Linux utilities and facilities that can enforce account security.

*Student background*: Students should know users, passwords, and authentication concepts; password configuration files; integrity checking of password files; and how to use the pluggable authentication module (PAM) to enhance security.

*Required facilities and software*: The PAM should be enabled.

*Explanation*: On completion of this exercise, students will understand

- the role and format of the password configuration files as they pertain to security (*/etc/passwd* and */etc/shadow* files);
- how to check the integrity of the */etc/passwd* and */etc/shadow* using the *pwck* utility;
- the use of the PAM to enhance password security; and
- other password management facilities that can enhance password security.

*Qualitative discussion*: Running the "*pwck"* utility does not always provide feedback to the student; students expect to see a message confirming that there is no problem with the */etc/password* file.

[1] http://www.safersite.com/pestinfo/j/john_the_ripper.asp
[2] http://northernlightsgroup.hypermart.net/nutcracker.html
[3] http://www.safersite.com/pestinfo/j/john_the_ripper.asp
[4] http://northernlightsgroup.hypermart.net/nutcracker.html

### Exercise Four: Users, Groups, and Security

*Pedagogical objective*: To defend a given Linux host by means of users and groups.

*Practical objective*: To understand how Linux groups can be used to enhance the security of a host.

*Student background*: Students should know users, groups, and security concepts; the importance of group planning; and granting access to certain groups of people.

*Required facilities and software*: "LiSt of Open Files" (Lsof) tool.

*Explanation*: Every UNIX/Linux user belongs to one or more groups. A group is a collection of users who share the same level of access and want to view and/or work with the data from other users in their group. To enhance security, system administrators can use groups to designate the sets of users who are allowed to read, write, and execute special files and directories, and grant access to certain devices. Adding users to groups requires careful planning on the part of the system administrator. System administrators must arrange users into groups, add the groups to the server, and then assign the users to the respective groups. On completion of this exercise, students should understand

- the importance of careful group planning;
- the process for creating user groups;
- how user private groups (UPGs) work. The idea behind UPGs is that it is more secure to have a user group with a single member. When an account is created, a single user group, referred to as a private group, which has the same name as the user, is created. This becomes the default group for all files created by that use; and
- how to grant access to a device to a certain group of users.

*Qualitative discussion***:** Students always question the importance of a group password if members can switch to the group with no password. It's always a surprise when students add themselves to the root group and are not given root privileges. With Red Hat 8.2, any user added to the wheel and root group automatically gets root privileges; this doesn't seem to work with Red Hat Linux 9.2.

### Exercise Five: Defending Accounts

*Pedagogical objective*: To make students aware of the potential computer security threats and alternative methods of prevention and detection.

*Practical objective*: To introduce students to the various ways attackers can gain entry to the system through accounts that are already in place.

*Student background*: Students should have knowledge of different approaches to defending accounts on a Linux host.

*Required facilities and software*: None.

*Explanation*: Every account that is set up on a system is a door to the outside that both authorized and unauthorized users can enter. Some accounts are well guarded, while others may not be. According to the 2005 Linux Security Resolution (http://www.linuxsecurity.com/ content/view/117721/49), a large number of corporate information security break-ins are due to stale user accounts. Students also use the *Lsof* tool to show open files. On completion of this exercise, students will know how to

- protect the superuser account. There are various ways to protect the root account, such as the following:
  - look for user identifiers (UIDs) of 0; a superuser has a UID of zero. Be suspicious of accounts that have a UID of 0, such accounts are usually

added by people who break into computers. Multiple superuser accounts are a bad idea and hard to control;

- restrict the number of people who can use the *su* command; the wheel group can protect the *su* command. If a malicious user can call *su* repeatedly, he or she will eventually be able to guess the root password. One can restrict the number of people who can use the *su* command by putting only few users into the wheel group;
- use the *sudo* utility to delegate some system responsibility to other people without giving away full root privileges. *Sudo* allows users to use passwords to access a limited set of commands. It also keeps a log of all successful *sudo* attempts, allowing one to track down those who used what command and what they did. For example, if a server needs to be administered by a number of people, it is usually not a good idea for all of them to use the root account. If they all log in with the same credentials, it becomes difficult to determine exactly who did what, when, and where. The *sudo* utility was designed to overcome this difficulty; it allows users defined in the */etc/sudoers* configuration file to have temporary access to run commands they would not normally be able to run due to restrictions in the permissions file. The following URLs are web sites with information on the *sudo* utility:
  - http://www.aplawrence.com/Basics/sudo.html
  - http://www.linuxhomenetworking.com/linux-hn/addusers.htm
- scan for root kits using *chkrootkit* [Lockhart 2004]. If one suspects that a system has been compromised, it is a good idea to check for root kits that may have been installed by an intruder. A root kit is a collection of programs that intruders often install after they have compromised the root account of a system; these programs help intruders erase their tracks;

- review accounts quarterly to verify the necessity for and validity of each account to ensure that none are left on the system by people no longer employed by the organization. Such accounts pose a significant security risk. As system administrators, students will check for
  - accounts without passwords: such accounts can compromise everything;
  - default accounts: computer systems are delivered to end-users with one or more default accounts. The accounts have standard passwords or no passwords. All default accounts that can be accessed by logging into them are a security vulnerability to the system; examples include *lp, sync, shutdown, halt, news, uucp,* and *operator* ;
  - accounts that run a single command: such accounts usually have no passwords; some include *date, uptime,* and *finger* ;
  - open accounts: some computer centers provide accounts on which visitors can play games; such accounts include *open*, *guest*, or *play*. These accounts do not usually require passwords; providing open accounts is a security risk;
  - dormant accounts: all dormant accounts must be deleted.
- use *Lsof*: this is a useful and powerful tool that will show open files. In UNIX, everything is a file: pipes, IP sockets, UNIX sockets, directories, devices, and

inodes. *Lsof* lists files opened by processes running on the system. Some of the tasks performed by students using *Lsof* include checking on the following:

- who is using a certain executable file, for example, the */etc/passwd* ?
- what files are opened on a device or who is accessing a certain drive?
- what files are opened by processes whose names start with, for example, *k* (*klogd*, *kswapd*...) and *bash*?
- what processes are opened by, for example, user apache and user johndoe?
- what files are using the process whose PID is 30297?
- what are the opened instances of directory */tmp* and all the files and directories it contains?

*Qualitative discussion*: Students take much longer to complete this exercise. The exercise puts together some defense mechanisms covered previously. The difference between the *sudo* and *su* utilities in Linux is often confusing to students from other disciplines.

### Exercise Six: File Permissions Management

*Pedagogical objective*: To defend a given Linux host through file permissions management.

*Practical objective*: To understand the security provisions of the Linux file system and how to restrict access to the files and directories to the file's owner, to a group of people, or to everybody using the computer system.

*Student background*: Students should have knowledge of file systems and security concepts; viewing access rights; the use of *chmod* and *umask* to enhance file security; how to keep track of the Set User Identifier (SUID) and the Set Group Identifier (SGID) files; and how to look out for world-writable files.

*Required facilities and software*: None.

*Explanation*: Any user who logs into the system with or without authorization poses a threat to system security. Such a user will be able to delete, create, and view files that might contain system-critical data. The file system is a primary tool for enforcing security in a Linux system. As system administrators, students must ensure that their system files are not open for casual editing by users and groups who should not be doing such system maintenance. On completion of this exercise, students will know how to

- view the access rights for a file using the *ls* command: It is essential to review critical file permissions. Well-managed access rights allow for a high level of security even without network security. Such a security scheme takes care of user access to programs and protects sensitive data such as home directories and system configuration files; hence, monitoring the permissions on system files is crucial to maintain host integrity;
- work with tools for file security, like the the *chmod* and *umask* utilities. Access rights will need to be changed for all types of reasons. Students use the *chmod* command to change access rights for files and directories. They also need to understand the meaning of the *umask* value; how to configure the user's file creation so that *umask* is as restrictive as possible, and how to compare the default file-creation umask value for the root account with that of other regular accounts. The root user account usually has stricter default file-creation permissions;

- keep track of the files that have the SUID and the SGID bit turned on, since intruders can create a SUID program as a backdoor into the system for their next visit;
- ensure that there are no world-writable files or directories on the system;
- identify hidden files, as they are a potential problem for UNIX systems. Any file that begins with dot (.) does not show up in the standard *ls* command; hackers have been known to use hidden files to hide their actions.

Once there is proper file permissions management, the advanced host-hardening techniques (logging and audit, encryption, and file integrity checking) can be applied.

*Qualitative discussion*: For this exercise to be meaningful, quite a bit more time has to be spent on its theoretical aspects, the use of the *chmod* and *umask* utilities, and file permissions. The two utilities can be confusing to students; however, spending time introducing Linux commands is not the most popular activity for computer majors.

### Exercise Seven: Integrity Management

*Pedagogical objective*: To make students aware of potential threats to computer security and alternative methods to prevent and detect them.

*Practical objective*: To teach students how to monitor the files system for unauthorized changes by using *Tripwire.*

*Student background*: Students should know integrity management concepts; understand how *Tripwire* works; be able to set up of *Tripwire* and analyze the results generated by *Tripwire*.

*Required facilities and software*: Tripwire.

*Explanation*: Students use *Tripwire*[5] to check the integrity of the files. *Tripwire* is a software security tool that works by comparing files and directories against a baseline. It first generates the baseline by taking a "snapshot" of specified files and directories in a known secure state. *Tripwire* then compares the current system against the baseline and reports any modifications, additions, or deletions. *Tripwire* is therefore useful for system security, intrusion detection, damage assessment, and recovery forensics. On completion of this exercise, students will know how to use *Tripwire* to

- generate policy and configure binary files with cryptographic signatures;
- create a baseline image of the entire network with a clean system image for each device and store it securely offline;
- run an initial integrity check;
- modify different files to see if *Tripwire* catches the changes, including adding a hidden file to the */etc/* directory, editing the */etc/hosts.deny* file, and editing the shadow file */etc/shadow* ;
- schedule systematic checks to detect any changes to the baseline; and
- generate a report to identify changes and interpret what has occurred; compare snapshots made at regular intervals to determine whether any critical part of the system has unexpectedly been altered.
- *Qualitative discussion*: *Tripwire* is straightforward to configure. Although the output reports are verbose, students are keen to find out their meaning and identify how their (the students') changes have been captured in the report.

[5] ftp://coast.cs.purdue.edu/pub/tools/unix/ids/tripwire/

## Exercise Eight: Auditing and Logging

*Pedagogical objective*: To defend the host through auditing and logging.

*Practical objective*: To understand the logging mechanism that Linux provides to help audit the usage and behavior of the system.

*Student background*: Students should know the auditing and logging concepts; understand the */etc/syslog.conf* file; the purpose of various logging files; how to track log files; and use the *logrotate* utility to enhance security.

*Required facilities and software*: *Logcheck*, a process accounting software, has to be set up.

*Explanation*: Whatever the protection mechanism established on the system is, it must be monitored. It is important to look for any indicators of misbehavior or other problems; hence, an intrusion-detection mechanism should be put in place. There are many ways in which monitoring can be done: two of the most common on Linux/Unix systems are spot inspections of file permissions (covered in Exercise Six) and the systematic review of Linux log files.[6] The process of monitoring the behavior of a system is called *auditing*, and an important part of it is system logging. Logging is one of the keys in keeping a system secure. The *syslogd* file keeps one informed on everything going on. System logging and file integrity checking can be used in combination to monitor the system for intruders, Although log file maintenance and log analysis tools can help catch potential intruders, they are not foolproof techniques [Barett et al. 2003]. Log files can be used to establish a recorded history of the events on a computer. On completion of this exercise, students will know how to

- plan for automated logging by exploring the contents of the */etc/syslog.conf*. As system administrators, students will plan what information is needed out of the system logs, figure out which logs contain the required information, and verify which log files are used by the system logging daemon, as shown by the fields in the */etc/syslog.conf* file;
- set up additional logging in the */etc/syslogd.conf* file;
- remotely log messages onto another computer;
- log user activity with process accounting, which allows one to keep detailed logs of every command that a user runs, including CPU time and memory used. Security-wise, the system administrator can gather information about what users ran, which command they used, and at what time. Such information is useful in discovering break-in or local root compromise, and can be used to spot malicious behavior by normal system users;
- use the *logrotate* command to help automate the compression and archiving of log files;
- explore the interpretation of individual log files with emphasis on security-related messages:
  - use the *lastlog* command to show the time a user logged in; if different, this might be a point of entry;
  - check the messages file for any break-in attempts and successes;
- track log files: the easiest way to view a log file is to search it using a command like *grep* or *tail*;

[6] Log files are important building blocks of a secure system. They form a recorded audit of the computer's past, making it easier to track attacks.

- use a log-monitoring tool: *logcheck. Logcheck* is a software package designed to automatically run and check system log files for security violations and all unusual activity. *Logcheck* uses a program called *logtail* that remembers the last position it read from a log file and will use this position on subsequent runs to process new information. *Logcheck* will, at 15-minute intervals, read the freshly appended part of the log files and *grep* it for any intrusions based on the keywords. In case of alerts, *Logcheck* immediately composes an email to the root (root@localhost) with a complete report of its findings.

The following are web sites with useful information on this subject:

- http://www.userlocal.com/security/seclogging.php
- http://www.freeos.com/articles/3540
- http://www.linuxjournal.com/article/5314

*Qualitative discussion***:** This exercise requires that the student has an email server set up and also have the account processing software.

### Exercise Nine: Cryptography Basics

*Pedagogical objective*: To defend a given Linux host.

*Practical objective*: To understand the roles of encryption and message digests in host security, and how Pretty Good Privacy (*PGP*) works.

*Student background*: Knowledge of cryptography basics concepts: encryption, decryption, digital signatures, and certificates.

*Required facilities and software*: PGP

*Explanation*: Information can be protected by using cryptography. The objective of this lab exercise is to introduce students to PGP, a tool that can be used to ensure the privacy of a message by encrypting it for its proper recipients. For this lab exercise, we used an exercise from the PGP tutorials: http://www.acm.org/crossroads/xrds6-5/pgptutorial.html. However, we extended it to include receiving and sending of encrypted mail.

*Qualitative discussion*: This exercise is more interesting if a key server is available on an isolated network; students can exchange keys through the key server.

### Exercise Ten: Research on a Security Tool

*Pedagogical objective*: To become aware of the "best of breed" tools and "best practices" in host security.

*Practical objective*: To expose students to the array of other security tools not covered in the course, that is, vulnerability assessment tools, intrusion detection, and forensics.

*Student background*: This is a capstone project; it assumes students understand the basics of host security in Linux. Students are free to choose either a Windows tool or a Linux tool.

*Required facilities and software*: Candidate tools include Sara, Rat, Nessus, Hunt, Ethereal, Squid, Satan, Protsentry, and Internet Security Scanner.

*Explanation*: This laboratory exercise is designed for a pair of students to work collaboratively on a security tool. On completion of this exercise, the students will have

- researched the tool;
- developed a tutorial for the tool;
- prepared and delivered a class presentation that includes a demonstration of how the tool works; and

- commented on their experience in using the tool, including ease of use, ease of installation, clarity of the documentation, and challenges encountered.

The pair project is expected to take three weeks of semester time; non-IT majors are paired with computer and MIS majors.

*Qualitative discussion*: This is the most interesting part of the course. Students are motivated to go out and find a tool, install it, and develop their own tutorial. They especially enjoy this part of the course because it gives them a sense of independence and responsibility.

## 4. ASSESSMENT, EXPERIENCES, AND CHALLENGES

The cybersecurity course has been offered for four years. When I began to teach the course, there were only two or three lab exercises, and most of the topics were theoretical. Evaluations for that year indicated the need for more lab exercises, so we sought funding to augment the existing materials. Student evaluations from the last few semesters have been positive: password-cracking, defending accounts, group security, *Tripwire,* and the research tool exercises were identified as the most interesting activities.

The research tool exercise was always interesting to computer science majors, but a challenge to some in other areas who had little or no technical background. I have always encouraged noncomputer science majors to pair up with computer science majors, but this is not always possible. Some students from nontechnical areas find the project a challenge due to the extensive documentation for some of the assigned tools, the challenging installation process, the fact that some of the tools require advanced Unix/Linux knowledge, and that the output from some tools is difficult to understand because it requires knowledge of basic network concepts. Less complex tools that non-computer science majors can work with are necessary for this exercise. Tools are assigned according to technical background.

Student evaluations also indicate that the exercises in the course helped them gain a better understanding of the concepts. Performance in examinations and understanding the concepts are closely correlated to some of the exercises. For example, the *Tripwire* exercise was identified as one of the most interesting: students were asked to assume a consultancy role to convince a client to use *Tripwire*. They generally began responding to this request by considering the purpose of *Tripwire*, explaining the most important files, their roles, and the configuration. Students have routinely done well in this exercise.

The quality of student feedback has been increasing with the continual improvement of the exercises. Although a questionnaire is currently being used to gather the impact of the exercises on student learning, there is need for a sounder student learning assessment technique.

During the first few weeks of the semester, nonmajors found Linux a challenge, and needed more time to complete the exercise and grasp the concepts. This is reflected in their performance on the first exam; however, there was a notable improvement by the second exam.

Work on the development of the Linux-based exercises continues. The challenges we have faced so far include identifying the ideal tools for teaching such a course to students from diverse backgrounds and integrating the laboratory exercises and the cybersecurity basics theories and principles. We realized that to integrate these two aspects of the

course in an effective and productive way, the content[7] and scope[8] of a cybersecurity basics course had to be defined first.

## 5. CONCLUSION

In the spring of 2004, in order to address the shortcoming noted above, we started work on defining the network security program standard (i.e., the scope and content of a network security program). We checked nonvendor-specific certifications such as the Global Information Assurance Certification (GIAC) offered by SysAdmin, Audit, Networking, and Security (SANS); the Certified Information Systems Security Professional CISSP) certification offered by ($ISC^2$); work from the Workshop on Education in Computer Security(WECS); SANS short courses, curriculum, and instruction literature [Smith and Ragan 1999]; major topics identified in 50 network security programs nationwide; and major topics covered in a widely used network security textbook [Pfleeger 1989]. We then developed a standard set of outcomes for a network security program. A paper based on this work was presented at the Pennsylvania Association of Computer and Information Science Educators Conference (PACISE) in April 2004 [Micco and Shumba 2004]. We believe that both the developed set of outcomes and similar work on identifying the common body of knowledge for information assurance courses [Davis and Dark 2003] can be a good basis for tool evaluation and a good guide for integrating the laboratory exercises and the information assurance course theories and concepts.

We are in the process of designing a new introductory course to operating systems, which will be a prerequisite for the cybersecurity basics course. It will introduce the basic Linux/Unix, and Windows operating systems to non-computer science majors.

As an extension of this work, we intend to include Windows-based exercises by using the Special Interest Group in Computer Science Education (SIGCSE) Special Projects grant that we recently received. The need for including Windows security tools into this course has also been highlighted in the student evaluations and by the department interns.

## REFERENCES

BARRETT, D.J., SILVERMAN, R.E., and BYRNES, R.G. 2003. *Linux Security CookBook*. 1st ed. O'Reilly Media, Sebastopol, CA.

DAVIS, J. and DARK, M. 2003. Defining a curriculum framework in information assurance and security. In *Staying in Tune with Engineering Education – Proceedings of the 2003 ASEE Annual Conference & Exposition* (Nashville, TN, June). http://www.asee.org/acPapers/2003-2281_Final.pdf.

GARFINKEL, S., SPAFFORD, G., and SCHWARTZ, A. 2003. *Practical UNIX & Internet Security*. 3rd ed. O'Reilly Media, Sebastopol, CA.

JACKSON, W. 2002. Cadets keep NSA crackers at bay. *Government Computer News* (May 20). http://www.gcn.com/print/21_11/18671-1.html

LOCKHART, A. 2004. *Network Security Hacks: 100 Industrial Tips and Tools,* 1st ed. O'Reilly Media, Sebastopol, CA.

MATETI, P. 2003. A laboratory-based course on internet security. In *Proceedings of the 34th Technical Symposium on Computer Science Education (SIGCSE 2003)*, ACM Press, New York, 252-256.

MICCO, M., and SHUMBA, R. 2004. Network security courses: Outcome based assessment. In *PACISE Pennsylvania Association of Computer and Information Science Educators Conference* (California University of PA, April 7, 2003).

MICCO, M., and ROSSMAN, H. 2002. Building a cyberwar lab: Lessons learned. In *Proceedings of the 33rd Technical Symposium on Computer Science Education (SIGCSE 2002)*, ACM Press, New York, 23-27.

PFLEEGER, C.P. 1989. *Security in Computing*. Prentice-Hall, Englewood Cliffs, NJ.

[7] Content represents the topics that should be taught; in other words, these are the core skills or the common body of knowledge for the course.

[8] Scope covers how much students should know and be able to do; this depends upon the degree of understanding/knowledge that the instructor intends for students to have upon completion of the course, and the key learning outcomes for each of the areas identified [Davis and Dark 2003].

SHUMBA, R. and MICCO, M. 2004. Towards a more effective way of teaching a cybersecurity basics course. In *Proceedings of the 35th Technical Symposium on Computer Science Education (SIGCSE 2004)*, ACM Press, New York, Poster presentation, p, 500.

SMITH, P.L. and RAGAN, T.J. 1999. *Instructional Design*. 2nd ed. Wiley, New York.

WAGNER, P. and WUDI, J. 2004. Designing and implementing a cyberwar laboratory exercise for a computer security course. In *Proceedings of the 35th Technical Symposium on Computer Science Education (SIGCSE 2004)*, ACM Press, New York, 402-406.

## APPENDIX

### Laboratory One -- Introductory Lab

**Activity One: Logging and Opening a Console**

1. Log into the machine using the provided username and password.
2. Look up and record the IP address and subnet address for the machine you are working on by entering: $*ifconfig*
3. Red Hat has a command and GUI interface. To be able to input some Linux commands, you need to open a console:
   a. Click on the "Red Hat" icon on the toolbar.
   b. Choose "Systems Tools" from the main menu, then "Terminal"
   c. A "Terminal" window opens; you are ready to type any commands.

**Activity Two: Giving Yourself Root Privileges**

To get root privileges:

$*/bin/su –*

This will prompt for a password.

To check who you are after changing your user ID, type

$*whoami*

To switch back from root, type

$*exit*

**Activity Three: Mounting Devices**

In Linux, before you can use a floppy disk or CD, you need to mount it. You need to be root to mount the devices.

There are two ways in which we can mount a device:

1. Use of the command line (console window )
2. Use of the "Disk management" tool

1. Use of the command line in the console window

   To mount a device, we use the mount command, which has the following pattern:

   $ *mount options device directory*

   a. To mount a floppy disk:
      $*mount /dev/fd0 /floppy*
   b. To mount a CD:
      $*mount /dev/cdrom /cdrom*
   c. To unmount your device, use the *umount* command:
      $*umount /dev/fd0*

2. Use of the disk management tool:
   a. Insert your disk/ CD
   b. Click on "System Tools" on the "Main Menu"
   c. Click on mount
   d. To list the files on your mounted device:
      $*ls /cdrom*

**Activity Four: Commonly Used Linux Commands**

NB: Each of the commands has an information file, which explains what the command does. To view information about any command and its options, we use the UNIX/ Linux command called *"man."* The *man* stands for manual pages.

1. To find out what directory you are working from right now (current working directory)? Type *pwd* at the prompt and record the directory.
   $*pwd*
2. To see the files and other directories in your current working directory, type *ls* at the prompt. Record your observation here.
   $*ls*. Record the names of any files
3. To create a directory within the directory you are working from (replace directory one with a directory of your choice).
   *$mkdir directoryone*
   To view the list of files in your current directory, including the one you have just created, type
   $*ls*
4. Now change your working directory to the one you have just created by typing *cd* at the prompt.
   $*cd directoryone*
5. Using the commands in step 3 above, create *directorytwo* in *directoryone.* Change your working directory to *directorytwo*. Record the command used for the two actions above.
6. We want to populate *directorytwo* with two or more empty files. To create the files *test* and *test1*, using the touch command.
   $*touch* test.
7. To make sure the file has been created, use the *ls* command.
8. Change your working directory to *directoryone*. To do that, type *cd* at the prompt, and then *pwd* to check if you have successfully changed the directory.
   $*cd..*
   *$pwd*
   - From wherever you are, the double dots *".."* in the *cd* command always refers to the parent directory, and the single dot *". "* always refers to the current directory. Thus, to move from your current working directory up to the respective parent directory, you can simply type *cd.*
   - Wherever you are, typing *cd* without an argument will bring you back to your home directory.
9. To remove the directory that you created last.
   $*rmdir directoryname*
10. To remove a file, type
    $ *rm trashfilename* or
    $ *delete trashfilename*.
11. We want to start searching for missing files using the *find* command. This is a very useful command if you are not sure in which folder a certain file is or have missed a file. Your search will take place in that directory and every directory downwards from it. We want to look for a file called "shadow".
    $*find -name shadow -print*

    a. To search for the same file starting at root
       $*find / -iname shadow -print*

b. To look for a directory
$*find -name youmissingdir -type d -print*

## Laboratory Two – Password Management I

**Activity One: Changing and Verifying Your Password**

To change your password, you DO NOT need to be root.

1. Log in with your regular account and change the password.
2. You need to verify your new password by logging into your account with the new password to make sure you have entered the password properly. You can do this without logging out. Use the */bin/su* command and your regular account username.
   $*/bin/su username*
   You will be prompted for the new password.
   Remember, if you are running as root, you can change the password of any other user, including yourself, without supplying the old password.

**Activity Two: Creating Accounts**

1. To add users, you need root privileges. Give yourself root privileges. Verify that you have root privileges before you continue.
2. There are two methods of creating accounts:
   a. the user and group tool, and
   b. the *useradd* command.

   Using method a), create five accounts where:
   a. password is a dictionary word (airport, floodgate, rupture)
   b. password is too short
   c. password is commonly used (baseball, lovebird, password)
   d. password is the same as username account
   e. password has a numeric suffix

   Using method b), create other five accounts where:
   a. password is repeating characters (e.g., 999999)
   b. password is absent
   c. password has a numeric prefix
   d. password has both uppercase and lowercase characters
   e. password takes two short words and combine them with a special character or number

   Methods a) and b) are described below. Use the provided table to fill in the details of the created user accounts. Use any usernames you prefer.

*Method a): Use of the user and group tool*

   a. Click on the "Main Menu"
   b. Select "System settings"
   c. Select "User and groups" option.
   d. Fill in all the fields
   e. Record the username, password, primary group for each created accounts in the table provided.

*Method b): useradd command*

   a. In the console window type
   $*useradd -g users -d /users/username -s /bin/bash -p password username*
   b. Explain each of the options given in the *useradd* command above, use the *man* command to get information about the *useradd* command.

**Activity Three: Auditing Existing Passwords**

1. Running *Nutcracker*
   *Nutcracker* is a simple dictionary cracker with options to running a hybrid attack.
   a. Give yourself root privileges.
   b. *Nutcracker* comes with a sample password file and a dictionary of words called *words*; you can test the provided password file against the dictionary.
   c. At the prompt type:
   $*/usr/sbin/nutcracker /etc/shadow /usr/share/dict/words*
   d. Record the results of running the *Nutcracker* program here. This may take hours. You may need to run this program out of class hours, since it takes a long time. Record any passwords cracked and the approximate time taken to crack the passwords.
   e. It is possible to specify options –p (X) for numerical prefixes, and s(X) for numerical suffixes. X is for digits from 0 to X, e.g., –p4 or –s9; -p or –s without a number checks 0 to 99
   $*/usr/local/sbin/nutcracker /etc/shadow /usr/share/dict/words –p -s*
   You need to be patient; this will take a bit of time to run. Record any cracked passwords and the time it takes.
2. Running *John the Ripper* (JTR)
   Now we will compare our results with another cracking program. *John the Ripper* is designed to be fast and powerful. It cracks standard and double-length DES, MD5 algorithms. You can suspend and restart a session. You can specify your own list of words. You can get the status of an interrupted session. You can specify which users or groups to crack.
   Give yourself root privileges, and follow the steps given below:
3. At the prompt type
   $./john /etc/shadow
4. Control C will suspend john.
5. To resume an interrupted session
   $*./john –restore*
6. To retrieve the cracked words
   $*./john –show /etc/shadow*
7. You can specify a list of words/dictionary to be used to try and crack the passwords.
   $*./john -wordfile: /usr/share/dict/words /etc/shadow*

Compare the results from the two crackers and document any interesting results.

**Activity Four: A Research Exercise on Cracker Programs**

1. How does a cracker program like *Nutcracker* work?
2. Pick another cracker program for Windows or UNIX or Linux, which has not been covered in this lab exercise. In no more than 2 pages
3. Explain the origin of the program, e.g. Author.
4. Environments under which it can be used.
5. How does it work?

## Laboratory Three – Password Management II

**Activity One: Understanding the etc/passwd and /etc/shadow Files**

Every user on the system must be able to read the */etc/passwd* file. However, the */etc/shadow* is only accessible to root.

1. Identify one entry of the accounts you have created in the last exercise and explain

the meaning of each of its fields in the */etc/passwd* file.

2. Now look at the */etc/shadow* entry for the same entry in 2. Explain the meaning of each of the fields in the */etc/shadow* entry.
3. What is the difference between the */etc/passwd* entries and */etc/shadow* entries? Explain security importance of the /etc/shadow file.
   It is always advisable to display the contents of the */etc/passwd* before you delete and make sure you are not deleting accounts with UID less than 500 because they belong to system services and might be critical for proper functioning of Red Hat.

**Activity Two: *Pwck* Utility for Checking the Integrity of Password File**

*Pwck*, is a system administration command that removes corrupt or duplicate entries in *the /etc/passwd* and */etc/shadow* files. *Pwck* verifies the integrity of the system authentication information. All entries in the */etc/passwd* and */etc/shadow* are checked to see that the entry has the proper format and valid data in each field. The user is prompted to delete entries that are improperly formatted or that have other in-correctable errors. Checks are made to verify that each entry has

a. the correct number of fields
b. a unique user name
c. a valid user and group identifier
d. a valid primary group
e. a valid home directory
f. a valid login shell

*Pwck* will prompt for a *yes* or *no* before deleting entries. You need to be root to run *pwck*.

1. To verify the integrity of the */etc/passwd* file.
   $*pwck*
2. This is one possible output from running the *pwck* command.

```
user adm: directory /var/adm does not exist
user news: directory /var/spool/news does not exist
user pcap: program /bin/nologin does not exist
pwck: no changes
```

3. Correct any noted problems. For any missing directories, create the directories**.** If no errors are displayed, then there are no problems with your *passwd* file. If any duplicate or corrupt passwords are encountered, *pwck* will report them. Explain any interesting results below:

**Activity Three: Use of PAM to Enhance Password Security**

PAM architecture was developed by Sun Microsystems. PAM is now used on virtually every UNIX/Linux system to provide improved user-level security, flexibility in managing user authentication, and smoother integration between Linux configuration data and user information stored on other systems. PAM modules can be edited to enforce user security. For example *pam_cracklib* can configure by default in Red Hat Linux to stop users from entering insecure passwords. To adjust password strength and other variables, you need to edit the */etc/pam.d/system-auth*

a. Locate the line on which the *pam_cracklib* module is referenced.
b. Change the *retry* parameter to *5*, the *type* parameter to *THE***,** and add a *minlen* parameter with a *value of 12*. The line should look like this:
   *password required /lib/security/pam_cracklib.so retry=5 type=THE minlen=12*
c. Locate the second line with the *pam_unix* module (of type password). Add the parameter *remember=15*. This causes the system to record the previous *15* passwords for each user so that they cannot be used again.

d. Try changing your password using various lengths and repeating previously used passwords to see how these settings affect what you can do.

**Activity Four: Other Facilities for Enhancing Security**

1. Pick one user account you created and lock it. This means no one can log in using that account. The password for the account remains intact. Check the /etc/shadow entry for locked account. Now unlock it. Perform the following tasks on that account:
   a. Set the minimum number of days that the current password can be used before it can be changed.
   b. Set the number of days before the password expiration date that the user will be warned (upon logging in) that the password must be changed to prevent the account from locking up.
   c. Set the number of days after an account is locked before it will become disabled (and its password deleted)
2. The password aging controls can be set in the */etc/login.defs*. Do a cat of the file and see if there are any settings you want to change.
3. As a system administrator, you can either create a locked account or lock an existing account. Let's try to understand the effect of creating a locked account with the *useradd* command:
   *$useradd Tim*
   a. Do a cat of the */etc/shadow* file. Note Tim's account. The two bangs after Tim indicate that the password hasn't been set for Tim's account. The account is locked.
   b. Unlock the account by issuing the *passwd* command to assign a password and set password aging guidelines:
   $*passwd Tim*

## Laboratory Four – User Groups and Security

**Activity One: Understanding Users, Groups and Security**

1. Every account that is set up on the system has a User ID and a Group ID. For each different username and group name, there is a unique UID or GID associated with it. Pick the root entry and two of your own created entries in the */etc/passwd* file and record the UID and the GIDs.
2. Each user belongs to a primary group, which is what is stored in the */etc/passwd* file. Find out the name of the group, and the other groups to which you belong.
3. List the groups to which another user belongs. You can use any accounts created before.
4. Red Hat Linux uses the notion of User Private Groups (UPG) for increased user security. This is how it works:
   a. When a user account is created, a group of the same name is also created.
   b. This group becomes the user's default, private group. The only member of the new group is the new user.
   c. Let's say you would like to have a group of people work on a set of files in the */usr/lib/emacs/site-lisp/* directory. You trust a few people to modify the directory but certainly not everyone.
      i. So first create an emacs group:
      $*/usr/sbin/groupadd emacs*
      ii. In order to associate the contents of the directory with the emacs group,

type:
$*chown -R root.emacs /usr/lib/emacs/site-lisp*

iii. Now, it is possible to add the proper users to the group with *gpasswd:*
$*/usr/bin/gpasswd -a <username> emacs*

iv. Allow the users to actually create files in the directory with the following command:
$*chmod 775 /usr/lib/emacs/site-lisp*

v. When a user creates a new file, it is assigned to the group of the user's default private group. To prevent this, perform the following command, which causes everything in the directory to be created with the emacs group:
$*chmod 2775 /usr/lib/emacs/site-lisp*

vi. If the new file needs to be mode 664 for another user in the emacs group to be able to edit it, make the default *umask 002*.

vii. At this point, by making the default *umask 002*, you can easily set up groups that users can take advantage of without any extra work, every time users write files to the group's common directory. Just create the group, add the users, and do the above *chown* and *chmod* on the group's directories.

5. Groups can also be used to allow certain users to exclusively access certain devices. This involves granting mount privileges to system users, creating the group, allowing the use of device by the group, and adding users to the group. This part of the tutorial is taken from this website:

   http://www.yolinux.com/TUTORIALS/LinuxTutorialManagingGroups.html

   a. Grant privileges to system users to mount the device. This requires a change to the file */etc/fstab*. The fourth column in the file defines mounting options. By default only root may mount the device (option owner). To grant users the ability to mount the device, change the owner option to user. With the user option, only the user who mounted the device can unmount the device. To allow anyone to unmount the device, use the option users.
   b. Create group cdrom.
   c. Allow use of device by group cdrom.
      $*chown owner:group* <device>
      For example: *chown root.cdrom /dev/hdd* **.**
   d. Allow group access to the device.
      $*chmod 660 /dev/hdd*
   e. Add user to group cdrom. At this point, adding users to the group cdrom will grant them access to the device.
      $usermod -G <comma separated group list> <user id>.

## Laboratory Five – Defending Accounts

**Activity One: Protect the Superuser Account**

1. **Check the */etc/passwd* file for UID of 0**. Record any such accounts. If there are users who do not deserve to be superusers, delete the UID of 0 in the */etc/passwd* for the respective entry.
2. **Restricting use of the *su* command**
   a. Record who owns the */bin/su* program.
   b. Add at least two users to the wheel group. Record the added users here.

c. Confirm that the two users are now members of the wheel group
d. Now, change ownership of the */bin/su* program to the wheel group.
e. Change the permissions for the */bin/su* program, so that the *su* command is protected from all normal users.
f. Pick a user who is a member of the wheel group and let that user try to use the *su* command. Record your observation.
g. Pick any other users who are not members of the wheel group and let them try to use the *su* command. Record any notable observation.
h. Using your answers from above, explain the relevance of the wheel group to security.

3. Use of the *sudo* command.
   a. Pick one of your created accounts and *su* to the account; e.g., jane's account
   b. As jane, try to view the contents of the */etc/sudoers* file (an action that normally requires privileged access). Record your observation**.**
   c. Now precede the viewing command used in b with the *sudo* command. Record any observation.
   d. Now view the contents of the */etc/sudoers* file, and check if the user is in the file. If they are not, please add them in and try the command again. If the user is in the */etc/sudoers* file, they are able to run the *sudo* command. Hints on the *sudo* command can be found at this website.
   http://www.aplawrence.com/Basics/sudo.html
   e. Now retry b and c above. Record your observation for each action.
4. Scan for root kits
   a. Run the *chkrootkit* from the directory it was built it.
   $*./chrootkit*
   b. Run *chckrootkit* on a mounted disk
   $*./chrootkit –p/mnt/cdrom*

**Activity Two: Check for Accounts Without Passwords**

1. You need to be root to read the */etc/shadow.*
   a. To search for accounts that do not have passwords, type the following:
   $*cat /etc/passwd | awk -F: 'length($2)<1 {print $1}'*
   List any accounts that do not have passwords**.**
   b. Now, to display entries for the accounts with no passwords, type one of the following:
   c. If only one account has no password:
   $*egrep 'nopasswd' /etc/passwd*
   d. If more than one account has no password, then separate each username with a | symbol.
   $*egrep 'username1|username2' /etc/passwd*
   e. Disable any such accounts.

**Activity Three: Check for Default Accounts**

Check your */etc/shadow* file for accounts that come with the system. Examples include *lp, sync, shutdown, halt, news, uucp, operator*. Once you have a list of such accounts, ensure that each is disabled or have a good password.

**Activity Four: Check for Accounts That Run a Single Command**

There are some accounts that simply run a single command. Such accounts usually have no passwords. Some of them include ____?. Look up these accounts that run a single

command. Examples are *date, uptime, and finger*.

**Activity Five: Check for Open Accounts**

Check for open accounts. Such accounts include open, guest, or play.

**Activity Six: Manage Dormant Accounts**

If there are any dormant accounts on your system, make sure they are disabled. To learn how to disable an account:

Pick three accounts you created and disable them using the methods described below. There are a number of ways ('a' to 'e' below) of disabling the accounts:

a. Lock the account.
b. Change the account login shell so that instead of letting the user type commands, the system simply prints informative messages and exits. In the password file, change the shell to */etc/disabled.* Record your action here.
c. Automatically disable an account if a user does not log in within a certain number of days. Record the command here**.**

**Activity Seven: Use of the *Lsof* Tool**

You need root privileges to run *lsof*.

This part of the exercise is from the modified tutorial on *lsof* given at this particular website:

*http://www.physiol.ox.ac.uk/Computing/Online_Documentation/lsof-quickstart.txt*

1. Run *lsof* without parameters, and record any listening or open files found
   $*lsof*
2. To display open ports with the related services,
   $*lsof -i*
   $*lsof -i -P*
3. (Put a screen shot from 2003)
   The above example shows that the ports 22 and 111, the port22 is used by sshd and the port 111 is used by portmap.
4. To know who is using a certain filesystem like */etc/passwd*, what files are opened on device */dev/hda6* or who is accessing a device like */dev/cdrom*[9]. Type the following and record your results:
   $*lsof /etc/passwd*
   $*lsof /dev/hda6*
   $*lsof /dev/cdrom*
5. To know which processes owns the TCP port 25, type the following command and record the results:
   $*lsof –i tcp:25*
6. To show the files are opened by processes whose names start by "courier", but exclude those whose owner is the user "karkoma":
   $*lsof -c courier -u ^karkoma*
7. To show me the processes opened by user apache and user johndoe:
   $*lsof -u apache,johndoe*
8. Search for all opened instances of directory */tmp* and all the files and directories it contains:
   $*lsof +D /tmp*

[9] To unmount a filesystem under UNIX, all the users must stop using it. However, it is often difficult to identify the user who has left a process running in that filesystem. *Lsof* allows the process still using any disk resource to be immediately identified.

9. Show me what files are opened by processes whose names starts by "k" (klogd, kswapd...) and bash. Show me what files are opened by *init*:
   $*lsof -c k*
   $*lsof -c bash*
   $*lsof -c init*

## Laboratory Six – File Attributes and Permissions

**Activity One: Viewing File Permissions**

The *ls* command is used to see what permissions are set for a particular file.
Record the permissions set on your home directory using BOTH symbolic and octal mode.

**Activity Two: Setting the File Permissions**

Setting permissions on files and directories is done using the *chmod* command. This document presents a simple explanation of *chmod.*

1. Create a directory called notes in your home directory. Populate the directory with at least two files. Record the permissions on the newly created directory.
2. Take away all the permissions from everyone, from the newly created directory except the read permission. Record the command used here.
3. Try changing to the newly created directory. Explain any observation**.**
4. Now, add the execute permission for the user. Record the command used here. Also check if you can now change to the created directory, and record your observation.
5. From the activities done above, what is the difference between setting a read access and setting a read and execute access on a directory?
6. Now lock your home directory so that no other users on the system can display its contents, but allows other users to access files or subdirectories contained within the directory if they know the names of the files or directories? Test this with your neighbor and record how you carried out the test.
7. You now want your neighbor to be able to add files and remove files from your home directory. Record what permissions you have to set. Test this out with your neighbor.

**Activity Three: SUID and SGID**

1. Find the SUID and SGID files on your system. If there are any located in user directories, investigate them to make sure that the system has not been compromised. You will need to pipe the output into a file using >.
   a. To find all of the SUID and SGID programs, type the following and record any two found files here
      $*find / -type f\( -perm -04000 –o –perm-02000 \) \-exec ls -lg {}*
   b. World-writable is another potential configuration flaw in UNIX. To find all world-writable files, type the following. Record any two found files here**.**
      $*find / -perm -2 –type f –ls*

**Activity Four: Look Out for Any Hidden Files**

$*find / -name '.*' –ls*

**Activity Five: *Umask* Value**

1. *Umask* is used to set default permissions. Display *umask* value and record it.
   $*umask*
2. Each UNIX has a system for creating files. Full permissions are granted to

everybody when creating a new directory. When creating a new file, the function will grant read and write permissions for everyone, but set the execute permissions to none. Before *umask* is applied, a directory has permissions 777 and plain files 666. The *umask* value is subtracted from the default permissions after the function has created the new or directories. If *umask* value is for example (0) 002, the default permissions is (777 -002) =775, for a file (666-002)-664.

a. Create a directory newdir and record the permissions in octal here.
b. Create a plain file and record the permissions in octal.
c. Note that the difference between the two octal values in **a** and **b** gives the *umask* value.

## Laboratory Seven -- File Integrity Management with *Tripwire*

**Activity One: Clean Up**

1. Change to the directory location of *Tripwire*, usually its */etc/tripwire* directory.
2. Using the wildcard '*' remove any files that start with the characters "my". Make sure you are in */etc/tripwire* before you delete.

**Activity Two: Review the Policy File**

You can specify how *Tripwire* software checks the system in the *Tripwire* policy file twpol.txt.

1. Make a copy of the original policy file. This is just a safety precaution.
   $*/usr/sbin/twadmin --print-polfile > mypol.txt*
2. Now, explore the contents of the policy file and edit it by typing your name at the top.
3. Save changes and print out the first page of the policy file to hand in.
4. Next, load the policy file back into *Tripwire*. (You must enter your site key passphrase to complete this command).
   $*/usr/sbin/twadmin --create-polfile mypol.txt*
   This command will update *tw.pol* policy file that *Tripwire* actually uses (You must enter your site key file passphrase to complete this command). Why is the site key file passphrase needed?

**Activity Three: Review the Config File**

You can specify how *Tripwire* software should check your system and where to find files by default in the *Tripwire* configuration file twcfg.txt. The configuration file should be reviewed and if necessary updated to fit this particular system.

1. First, make a copy of the original configuration file.
   $*/usr/sbin/twadmin --print-cfgfile > myconfig.txt*
   *myconfig.txt* is your new config file, to play around with.
2. Open the config file for editing.
   $*gedit myconfig.txt*
3. Change/Add the following fields:
   a. EDITOR = /usr/bin/pico
   b. LATE PROMPTING = true
   c. LOOSE DIRECTORY CHECKING = true
   d. RESETACCESSTIME = true
4. Save and exit the editor.
5. Reload the config file back into *Tripwire*.
   $*/usr/sbin/twadmin --create-cfgfile --site-keyfile site.key myconfig.txt*

The site key file passphrase will be required.
6. To conclude this activity, print the *myconfig.txt* for handing in**.**

**Activity Four: Creating the Initial Database**

1. With the configuration and policy files set up to our preferences, it is now time to create the initial checksums for every important file on the system in a database.
   $*/usr/sbin/tripwire --init --verbose*
2. Enter your localphrase and wait for the database to be created.
3. After the database has been created, the following message will be displayed:
   *Wrote database file: /var/lib/tripwire/labxx.cyberlabx.net.twd*
4. Now run an integrity check:
   $*/usr/sbin/tripwire --check > initialcheck*
5. Enter the site key passphrase to complete this command.
6. What is the purpose of the initial database file we have just created?

**Activity Five: Checking the Integrity of Our Files**

1. Modify different files to see if *Tripwire* will catch the changes.
   a. Add a hidden file to the *etc* directory
      i. $*cd /etc*
      ii. $*touch .myhiddenfile*
   b. Edit the */etc/hosts.deny* file and add an entry for 192.168.44.22
   c. Make a change to the system configuration files by adding a new user (thus altering /*etc/passwd*, a critical configuration file)
   d. Edit the shadow file, */etc/shadow*, and change the user and group id for a user
2. Now run *Tripwire* again to check the current checksums against the database:
   $*/usr/sbin/tripwire --check > secondcheck*
3. Open the results in a text editor. Can you locate the modification you did above?

**Activity Six: Reviewing the Report**

1. Take a look at the summary of the report:
   $*/usr/sbin/tripwire --check --interactive*
   a. A number of files will be reported as non-existent. There is need to clean up the policy file to get rid of entries for files you do not have.
   b. For the rest of this lab, work on cleaning up the non-existent files. Delete the policies in the policy file as needed. List at least three policies you have identified for deleting.
2. After running a check, determine that the alterations in your system are valid--they were made by an authorized system administrator. Then, update the *Tripwire* database so that those changes don't appear in the next report. To do this, determine the name of the database file in the directory */var/lib/tripwire*, and then include the filename in this command.
   $*/usr/sbin/tripwire –update –d filename*
3. To review the results database that was created, you can use the *twprint* command. To use the *twprint* command, the full name of the report is required. The report name can be obtained by going to the */var/lib/tripwire/report* folder and looking it up.
   $*twprint –m r –twrfile /var/lib/tripwire/report/ ___.twr*

## Laboratory Eight – Logging and Auditing Lab

**Activity One: Exploring */etc/syslog.conf***

1. Open the */etc/syslogd.conf* to see exactly which log files are used by the system

logging daemons. List the files used by any four daemons.

2. Additional logging information can be set up in the */etc/syslogd.conf*. Add the following lines to the bottom of the */etc/syslogd.conf* file. Please make sure whatever *ttys* listed in the additional information are available.
   **.* /dev/tty2*
   **.warn;authpriv.* /dev/tty11*
3. To forward all logging traffic from a machine to your central log host, do the following:
   a. Add the following to the /etc/syslogd.conf
      **.* @loghost*
   b. Add an entry to */etc/hosts* for the "loghost"
      *192.168.136.3 loghost loghost.akadia.ch*
   c. On the loghost, you have to tell *syslogd* to accept messages from the network by adding the –r option:
      $*syslogd –r*

It is possible to have line (a) as the only line in your */etc/syslogd*. In this case, messages will be logged to the remote host, if added to what's already there, the logs are stored both locally and remotely.

For security purposes, keep an eye on what's written under the auth facility. Multiple login failures can indicate an attempted break-in.

**Activity Two: Logging User Activity with Process Accounting**

1. To enable process accounting, run a startup script to enable the process:
   $*chkconfig acc on*
   $*/sbin/service acct start*
   The process accounting package provides several programs to make use of the data that is being logged.
2. To print out the number of hours logged by the current user, type
   $*ac*
3. To search accounting logs by username and command, type the commands below and record results.
   $*lastcom andrew*
   $*lastcom bash*
4. To list all the commands found in the accounting logs and to print the number of times that each command has been executed, type the following:
   $*sa*
   To output the per user statistics:
   $*sa –u*

As system administrators, peruse the output of these commands every so often to look for suspicious activity, such as commands that are known to be used for mischief, and increase in CPU usage.

**Activity Three: Rotating Log Files**

1. If you are running Red Hat, you almost certainly have *logrotate* running. In fact, you should see an entry for it in your */etc/cron.daily* directory. This is a simple script that calls *logrotate* with the default configuration, at */etc/logrotate.conf*. The next step is to edit */etc/logrotate.conf*, which defines what *logrotate* does and how. Configuration parameters exist in both a global configuration file and one for each subsystem. The global file is (by default) */etc/logrotate.conf* while the subsystem specific definitions are in the directory */etc/logrotate.d*. Look up the definitions in the */etc/logrotate.conf*

file and explain them.

2. Change the configuration of your system to rotate log files every day and to save log files for seven days instead of the default (normally four). You can review the *man* page for *logrotate* if you are unsure what keywords to use for these changes). Record what changes you had to make.

**Activity Four: Permissions on Log Files**

1. Restrict access to log directory and *syslogd* files for normal users by using the following:
   $*chmod 751 /var/log /etc/logrotate.d*
   $*chmod 640 /etc/syslogd.conf /etc/logrotate.conf*
   $*chmod 640 /var/log/*log*
   $*chmod 644 /var/log/wtmp /var/log/utmp*
2. The */var/log/wtmp* and */var/log/utmp* files contain login records for all users on the system. Their integrity must be maintained because they can be used to determine when and from where a user has entered the system.

**Activity Five: Use of *Logcheck* Tool**

*Logcheck* does more than simply display log entries. It checks them hourly for an suspicious entries. If so, those entries are emailed to the root user.

1. Review the contents of */etc/logcheck* to better understand the types of attacks that *logcheck* is watching for. Record the types of attacks here.
2. Log out of your system, and try to re-log in with a wrong password. After successfully logging in, give yourself root privileges. Check your email messages by typing *mail* at the command line. Record the latest message; try to understand what the message is saying. Record all the commands you have used here.

Received March 2005; revised January 2007; accepted January 2007