ETHICAL HACKING
LAB SERIES

Lab 15: Understanding Buffer Overflows

Material in this Lab Aligns to the Following Certification Domains/Objectives		
Certified Ethical Hacking (CEH) Domains	Offensive Security (PWK) Objectives	SANS GPEN Objectives
17: Buffer Overflow	7: Buffer Overflows	5: Exploitation Fundamentals

Document Version: 2016-03-09

Contents

Introduction

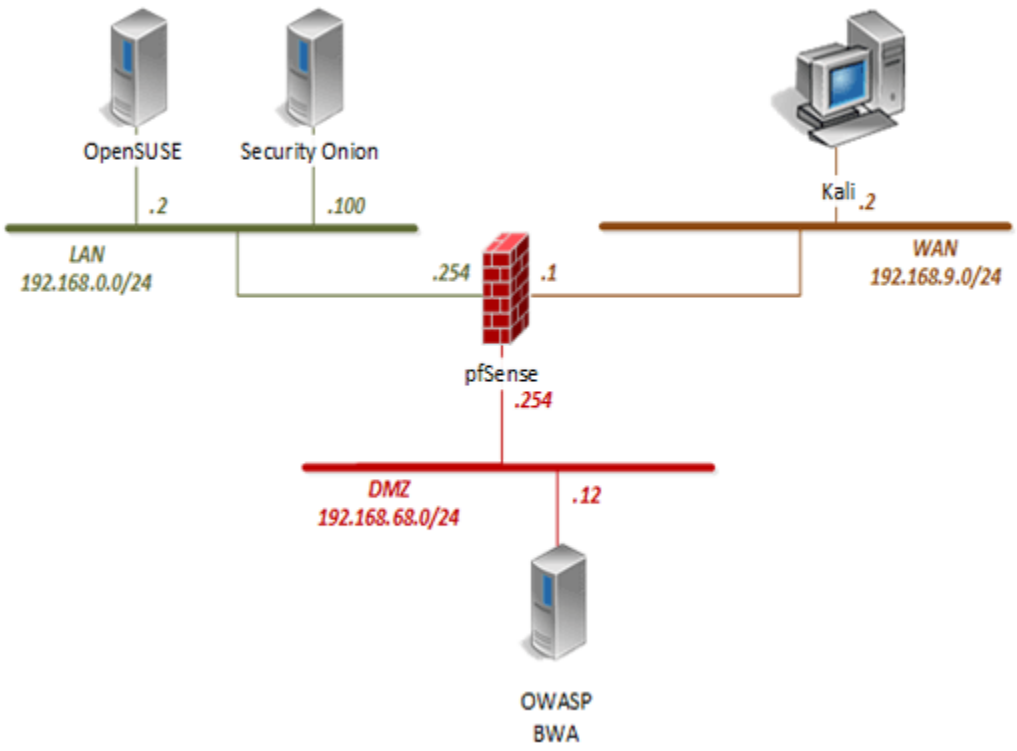
Buffer overflows are programming errors whether intentional or accidental. This lab shows how to create a vulnerable program and demonstrate the vulnerability.

Objective

In this lab, you will be conducting ethical hacking practices using various tools. You will be performing the following tasks:

1. Writing a Buffer Overflow Program
2. Run Code to Demonstrate Buffer Overflow
3. Analyzing and Modifying Overflow Code

Pod Topology



Lab Settings

The information in the table below will be needed in order to complete the lab. The task sections below provide details on the use of this information.

Virtual Machine	IP Address	Account (if needed)	Password (if needed)
Kali Linux	192.168.9.2	root	toor
pfSense	192.168.0.254	admin	pfsense
OWASP Broken Web App	192.168.68.12	root	owaspbwa
OpenSUSE	192.168.0.2	osboxes	osboxes.org
Security Onion	n/a	ndg	password123

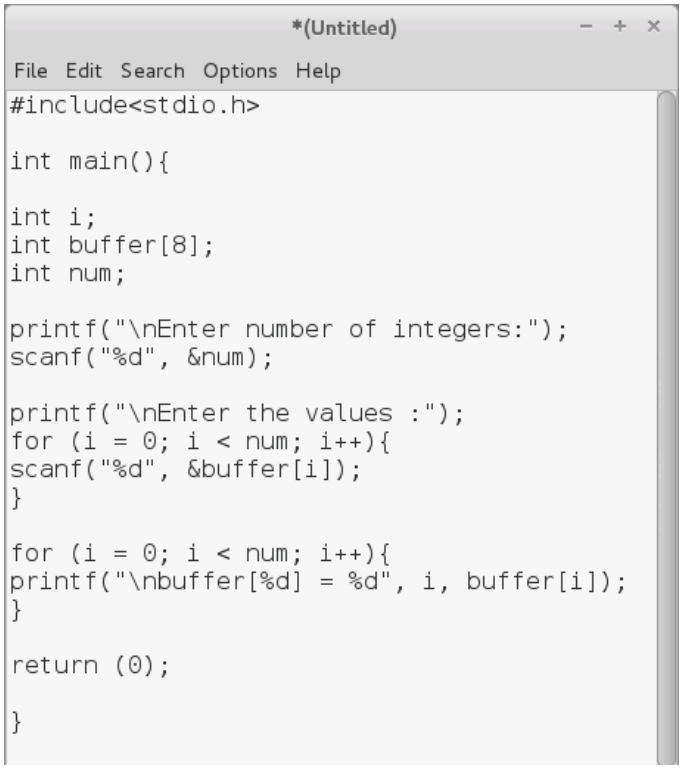
1 Writing a Buffer Overflow Program

1. Navigate to the *topology* page and click on the **Kali** VM icon.
2. Click anywhere within the *Kali* console window and press **Enter** to display the login prompt.
3. Enter **`root`** as the *username*. Click **Next**.
4. Enter **`toor`** as the *password*. Click **Sign In**.
5. Launch the *Leafpad* editor by clicking on the **Leafpad** icon in the left panel.

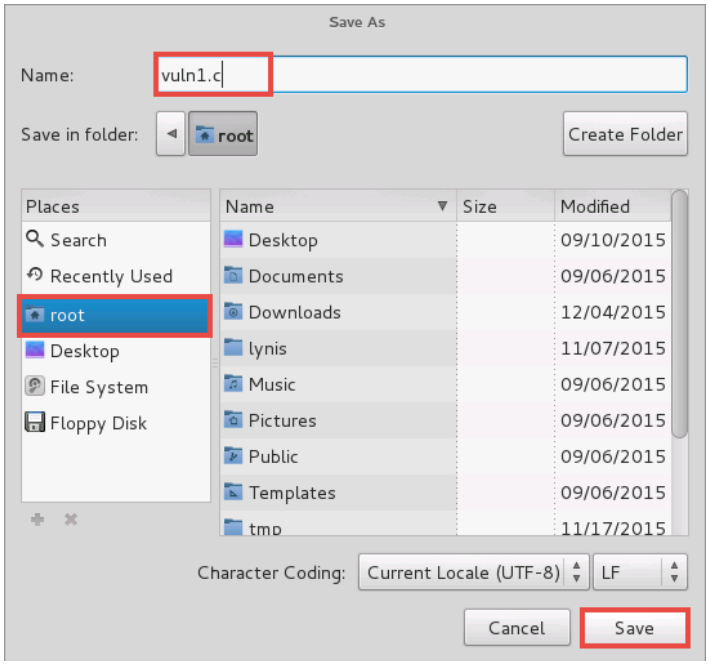
6. Using the *Leafpad* editor, type the script below. The red font are comments and they only explain what each respective line processes and does not need to be included in the script.

```
#include<stdio.h>

int main(){

int i;              //integer value for variable "i"
int buffer[8];      //an array of integers called "buffer" of up to 8 chars
int num;            //integer value for variable "num"

printf("\nEnter number of integers:");  //get user's input on the # of ints
scanf("%d", &num);                      //place values into the "num" variable

//Read into array section
printf("\nEnter the values :");   //get the integers from the user
for (i = 0; i < num; i++){        //loop to read the integers into the "buffer"
scanf("%d", &buffer[i]);
}

//Print the array of integers
for (i = 0; i < num; i++){              //loop thru buffer and print integers
printf("\nbuffer[%d] = %d", i, buffer[i]);
}

return (0);         //end the program

}
```


Text should read as shown below:



7. Click the **File** menu option from the top pane and select **Save As**.
8. Click on the **root** home directory on the left pane under *Places*.
9. Type **`vuln1.c`** into the *Name* text field.
10. Click **Save**.



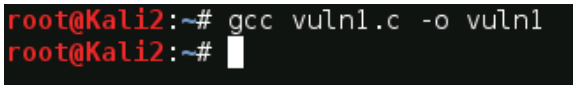
11. Close the **Leafpad** application.

2 Run Code to Demonstrate Buffer Overflow

1. Open the *Terminal* by clicking on the **Terminal** icon located on the left panel.

2. Compile the *C* code by typing the command below into the *Terminal* followed by pressing the **Enter** key.

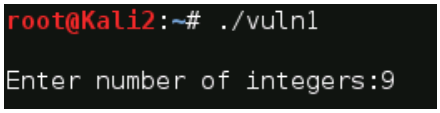
```
gcc vuln1.c –o vuln1
```



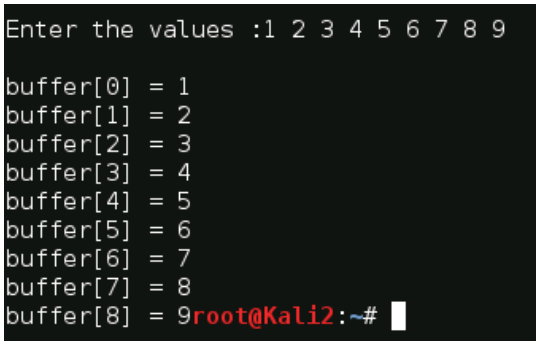
3. Run the program by entering the command below.

```
./vuln1
```

4. When prompted for the number of integers, type **9** even though the buffer only allocates 8. Press **Enter**.



5. When prompted to enter the values, type `1 2 3 4 5 6 7 8 9`. Press **Enter**.



Notice a buffer overrun was caused because 9 elements were chosen as an input into a buffer of 8.

3 Analyzing and Modifying Overflow Code

1. Open the **Leafpad** application once more from the left panel.
2. Click **File** and choose **Open**.
3. Select **vuln1.c** from the middle pane and click **Open**.
4. Observe the contents of the vuln1.c program:

```
#include<stdio.h>

int main(){

int i;
int buffer[8];
int num;

printf("\nEnter number of integers:");
scanf("%d", &num);

printf("\nEnter the values :");
for (i = 0; i < num; i++){
scanf("%d", &buffer[i]);
}

for (i = 0; i < num; i++){
printf("\nbuffer[%d] = %d", i, buffer[i]);
}

return (0);

}
```

A

B

C

D

E

a. *Section A*: This sets up the variables called “i”, “num”, and an array or list of characters “buffer” of up to 8 in length.

b. *Section B*: This section asks the user how many integers to put into the array. Note that the “scanf” function is the problem.

c. *Section C*: Asks the user for the integers to use and then puts them one by one into the array “buffer”.

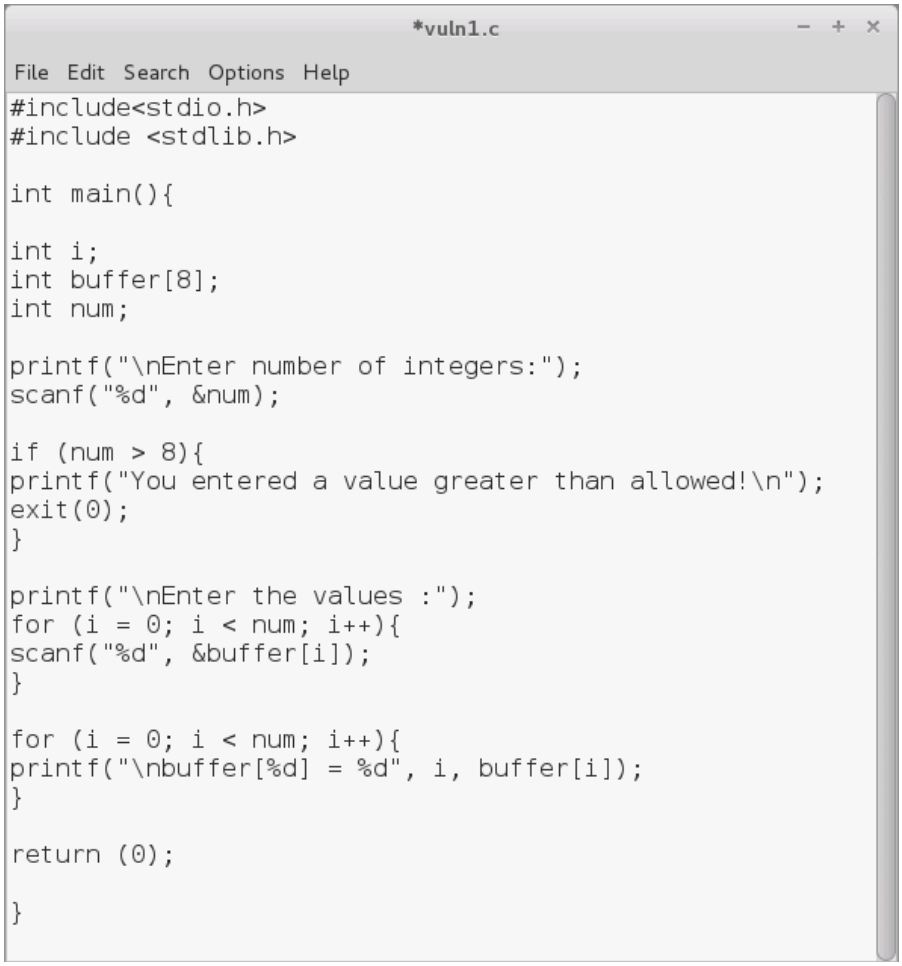
d. *Section D*: Prints out the integers that the user inputted previously.

e. *Section E*: Ends the program.

5. Alter the code to prevent an overflow. Add the following lines of code to *vuln1.c* so it reads as shown below.

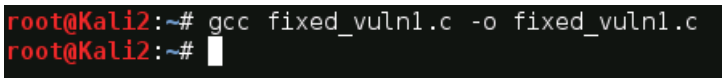
```
#include<stdio.h>
#include <stdlib.h>

int main(){

int i;              //integer value for variable "i"
int buffer[8];      //an array of integers called "buffer" of up to 8 chars
int num;            //integer value for variable "num"

printf("\nEnter number of integers:");  //get user's input on the # of ints
scanf("%d", &num);                      //place values into the "num" variable

if (num > 8){              //check the input is less than 8 numbers
printf("You entered a value greater than allowed!\n");
exit(0);                   //exit the program
}

//Read into array section
printf("\nEnter the values :");   //get the integers from the user
for (i = 0; i < num; i++){        //loop to read the integers into the "buffer"
scanf("%d", &buffer[i]);
}

//Print the array of integers
for (i = 0; i < num; i++){              //loop thru buffer and print integers
printf("\nbuffer[%d] = %d", i, buffer[i]);
}

return (0);         //end the program

}
```


The text should read as below:



6. Click **File** and select **Save As**.
7. Click on the **root** home directory on the left pane under *Places*.
8. Type **`fixed_vuln1.c`** into the *Name* text field.
9. Click **Save**.
10. Close the **Leafpad** application.
11. Change focus to the **Terminal** window and enter the command below to compile the code for the new *fixed_vuln1.c* program.

```
gcc fixed_vuln1.c –o fixed_vuln1.c
```



12. Run the new code.

```
./fixed_vuln1.c
```


13. When prompted for the number of integers, type **9**. Press **Enter**.

Notice an output is given signaling that a value greater than 8 was entered and is not allowed resulting in the program to exit.

14. Close the **Kali** PC viewer.

