NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

INFORMATION SECURITY LAB

NAME	Ayesha Imran
Class	CS-A
Lab	09
Course	Information Security
Date	24-November-25
Submitted To	Lec. Attiya Ashraf

LAB TASKS

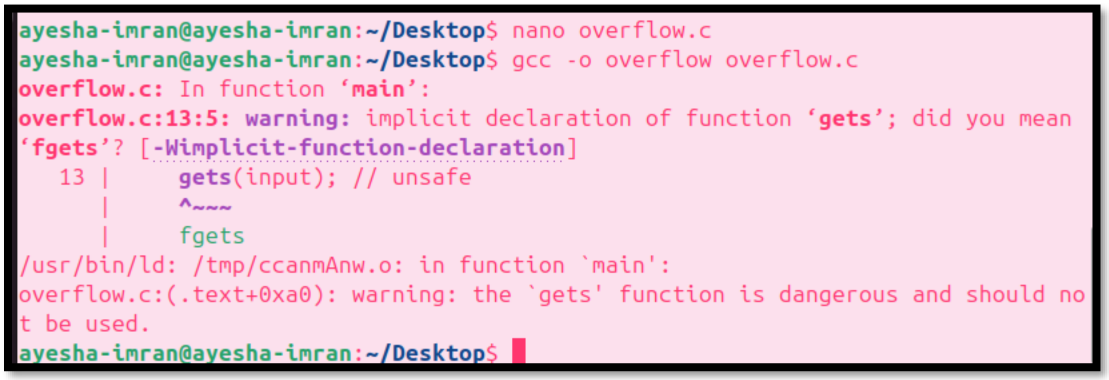
2.1 Compile and Run the Program:

1. Create the C program:

o Open a text editor (e.g., Nano or Vim), copy the above code, and save it as buffer_overflow.c.

2. Compile the Program:

o Open a terminal and run: gcc -o overflow buffer_overflow.c

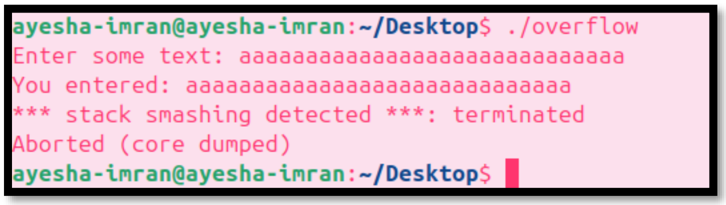


```
ayesha-imran@ayesha-imran:~/Desktop$ nano overflow.c
ayesha-imran@ayesha-imran:~/Desktop$ gcc -o overflow overflow.c
overflow.c: In function ‘main’:
overflow.c:13:5: warning: implicit declaration of function ‘gets’; did you mean
‘fgets’? [-Wimplicit-function-declaration]
   13 |     gets(input); // unsafe
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/ccanmAnw.o: in function `main':
overflow.c:(.text+0xa0): warning: the `gets' function is dangerous and should no
t be used.
ayesha-imran@ayesha-imran:~/Desktop$
```

3. Run the Program:

o Execute the program::

Command: ./overflow



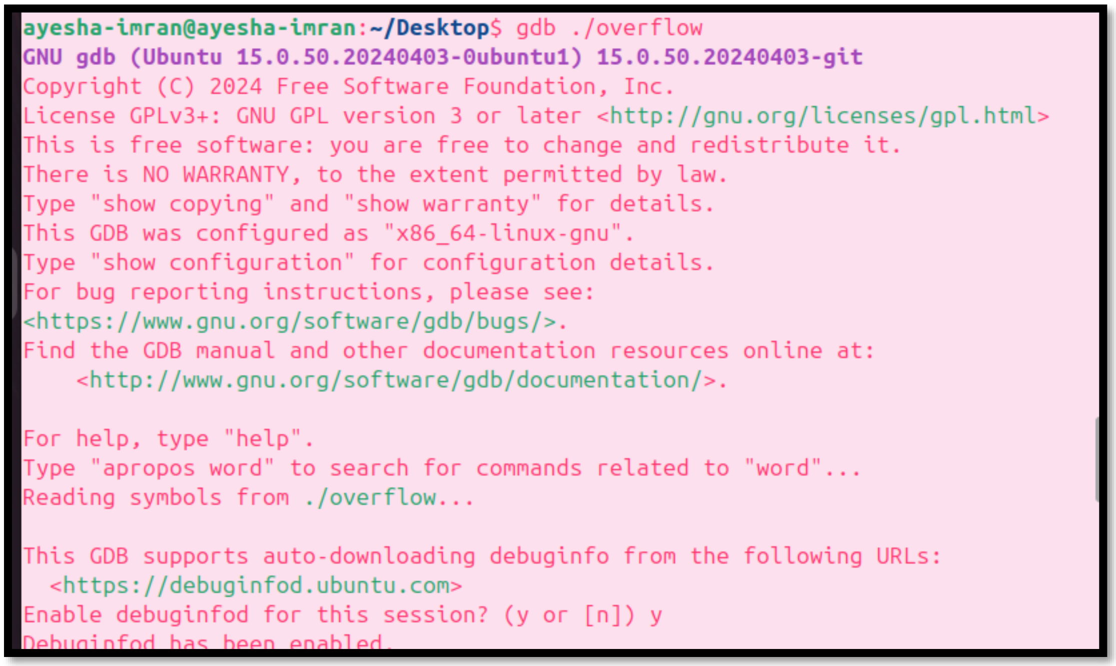
```
ayesha-imran@ayesha-imran:~/Desktop$ ./overflow
Enter some text: aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
You entered: aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: terminated
Aborted (core dumped)
ayesha-imran@ayesha-imran:~/Desktop$
```

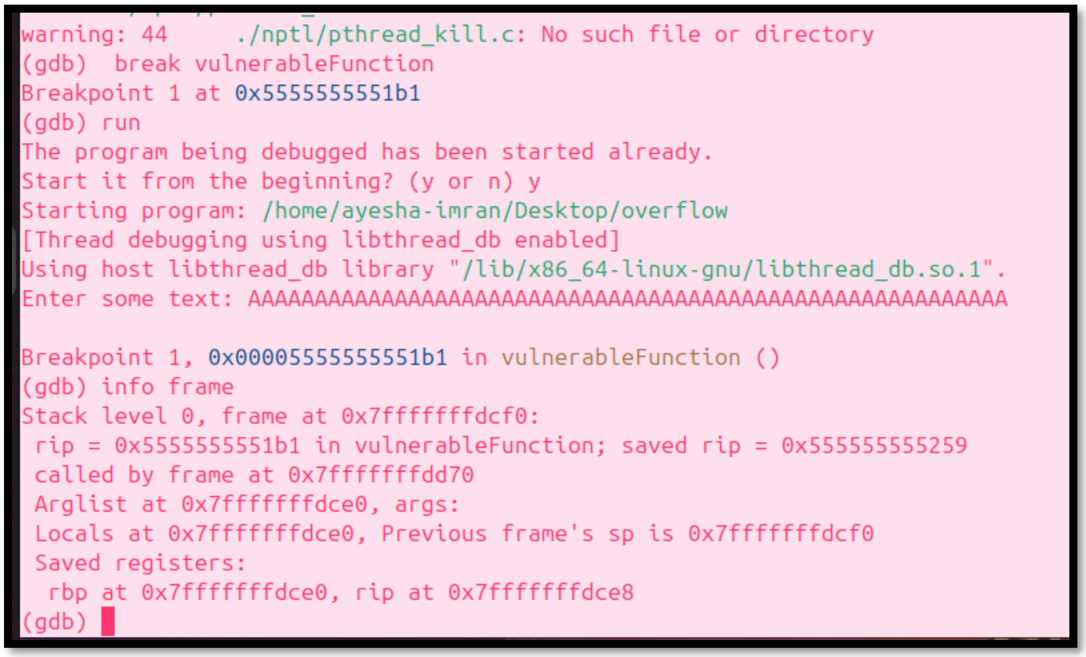
2.3 Inspect Memory and Stack:

1. Use gdb to Debug:

o To analyze the program's behavior in more detail, use the GNU Debugger (gdb):

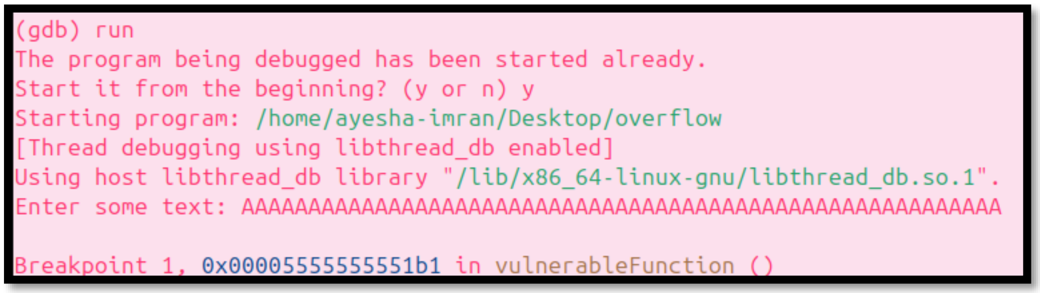
Command: gdb ./overflow


```
ayesha-imran@ayesha-imran:~/Desktop$ gdb ./overflow
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
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-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 ./overflow...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
```


```
warning: 44     ./nptl/pthread_kill.c: No such file or directory
(gdb)  break vulnerableFunction
Breakpoint 1 at 0x5555555551b1
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ayesha-imran/Desktop/overflow
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter some text: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, 0x00005555555551b1 in vulnerableFunction ()
(gdb) info frame
Stack level 0, frame at 0x7fffffffdcf0:
 rip = 0x5555555551b1 in vulnerableFunction; saved rip = 0x555555555259
 called by frame at 0x7fffffffdd70
 Arglist at 0x7fffffffdce0, args:
 Locals at 0x7fffffffdce0, Previous frame's sp is 0x7fffffffdcf0
 Saved registers:
  rbp at 0x7fffffffdce0, rip at 0x7fffffffdce8
(gdb)
```

3. Run the Program:

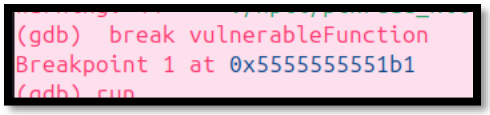
o Execute the program: (gdb)run



```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ayesha-imran/Desktop/overflow
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter some text: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, 0x00005555555551b1 in vulnerableFunction ()
```

2. Set a Breakpoint:

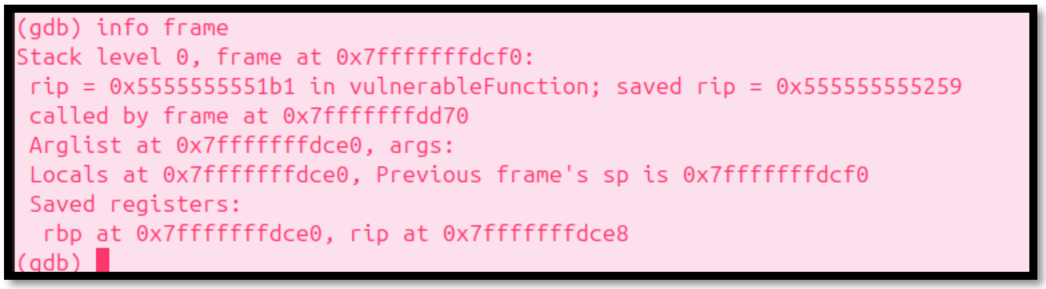
o Set a breakpoint at the vulnerable function:



```
(gdb)  break vulnerableFunction
Breakpoint 1 at 0x5555555551b1
(gdb) run
```

3. Analyze the Stack:

o When the program hits the breakpoint, use the following command to inspect the stack:

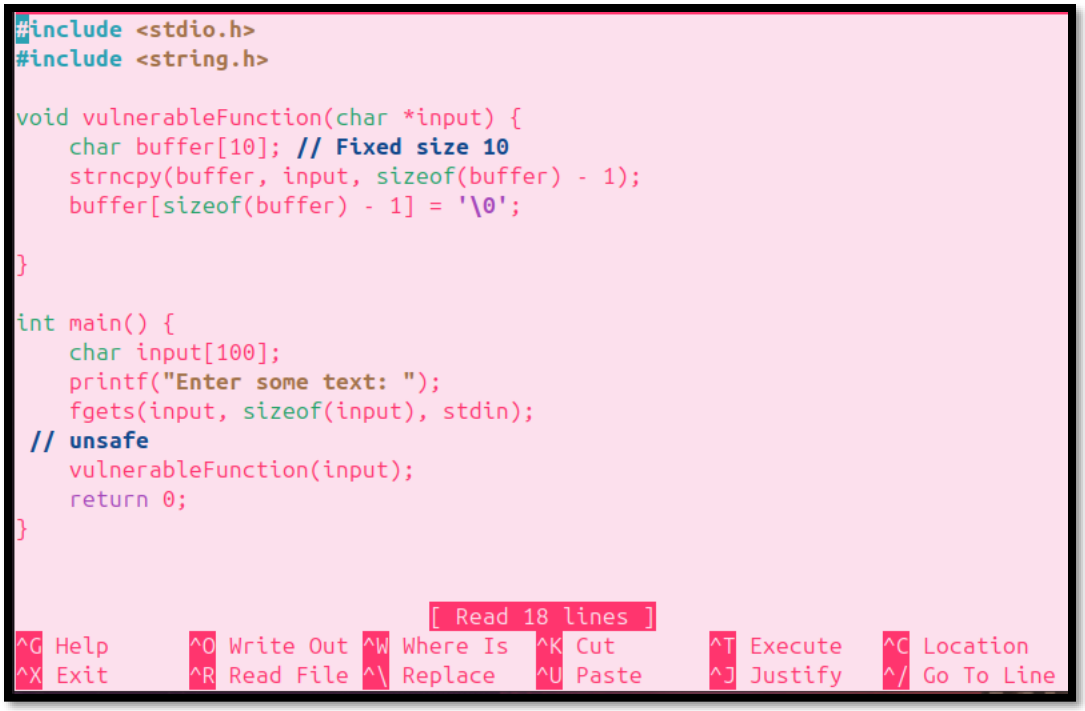


```
(gdb) info frame
Stack level 0, frame at 0x7fffffffdcf0:
 rip = 0x5555555551b1 in vulnerableFunction; saved rip = 0x555555555259
 called by frame at 0x7fffffffdd70
 Arglist at 0x7fffffffdce0, args:
 Locals at 0x7fffffffdce0, Previous frame's sp is 0x7fffffffdcf0
 Saved registers:
  rbp at 0x7fffffffdce0, rip at 0x7fffffffdce8
(gdb) 
```

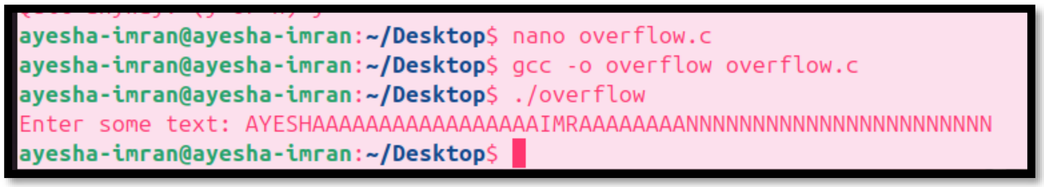
a) Part 3: Mitigation and Prevention

b) 3.1 Fix the Vulnerability in the Code

First fix the code



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

void vulnerableFunction(char *input) {
    char buffer[10]; // Fixed size 10
    strncpy(buffer, input, sizeof(buffer) - 1);
    buffer[sizeof(buffer) - 1] = '\0';

}

int main() {
    char input[100];
    printf("Enter some text: ");
    fgets(input, sizeof(input), stdin);
 // unsafe
    vulnerableFunction(input);
    return 0;
}
```



```
ayesha-imran@ayesha-imran:~/Desktop$ nano overflow.c
ayesha-imran@ayesha-imran:~/Desktop$ gcc -o overflow overflow.c
ayesha-imran@ayesha-imran:~/Desktop$ ./overflow
Enter some text: AYESHAAAAAAAAAAAAAAAAAIMRAAAAAAAAANNNNNNNNNNNNNNNNNNNNN
ayesha-imran@ayesha-imran:~/Desktop$ 
```

Analysis of Logs: Program Crashing Due to Buffer Overflow

Observations

- **Input provided:** A long string of repeated As (e.g., 40–100 characters).
- **Program behavior:**
    - Printed part of the input.
    - Triggered **stack smashing detected** or **Segmentation fault**.
    - Execution aborted by the operating system.

Analysis

- The buffer (char buffer[10]) was only 10 bytes long.
- The unsafe strcpy() copied the entire input without bounds checking.
- This overwrote adjacent memory, corrupting the stack frame.
- The compiler's stack protector detected the corruption and aborted the program.

- This demonstrates how buffer overflow can lead to program crashes and potential exploitation.

3. Analysis of Logs: Program After Vulnerability is Fixed

Observations

- **Input provided:** Same long string of repeated As (≥ 40 characters).
- **Program behavior:**
    - Printed a truncated version of the input.
    - No crash occurred.
    - Program exited gracefully.

Analysis

- The fixed code replaced strcpy() with strncpy(), limiting the copy to buffer size minus one.
- The buffer was explicitly null-terminated (buffer[sizeof(buffer)-1] = '\0').
- gets() was replaced with fgets(), ensuring input length is bounded.
- As a result, even when long input is provided, the buffer only stores up to 9 characters plus the null terminator.
- The program now handles input safely, preventing overflow and segmentation faults.