

Internet Security – COS80013 Lab - 3 Report

Student ID: 104837257

Student Name: Arun Ragavendhar Arunachalam Palaniyappan

Lab Name: COS80013 Lab 3 – Buffer Overflow in C

Lab Date: 26 /03/2025

Tutor: Yasas Akurudda Liyanage Don

Title and Introduction

This lab was about understanding **buffer overflow vulnerabilities** in C. Using three programs (memtest1.c, memtest2.c, and safegets.c), the learning from the lab was , how unchecked input can corrupt memory, cause unexpected behaviour, and crash programs. The aim was to explore how attackers exploit such bugs and how developers can prevent them using safer coding practices.

Methodology

1. memtest1.c

- Compiled and ran the program.
- Input strings of increasing length into a small buffer (char buf[8]).
- Observed when the variable i—located near the buffer—started getting corrupted.
- Calculated the offset required to trigger the overflow and induce a crash.

2. memtest2.c

- Used gets() to input first and last names (both char[12]).
- Entered long input into last, causing it to overwrite first.
- Resulted in memory corruption and overwriting of first name with the overflowed part of the second name.

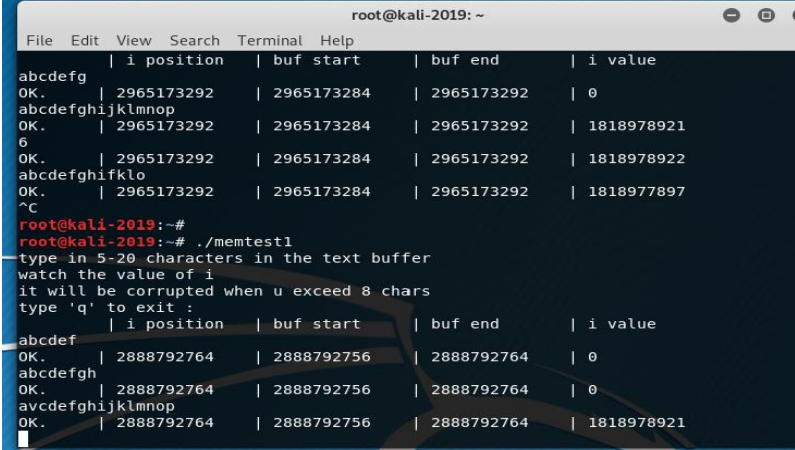
3. safegets.c

- Used fgets() with a helper fixgets() function to safely handle input.
- Verified that overflows were prevented, and newlines were properly trimmed.

Data Recording and Observations:

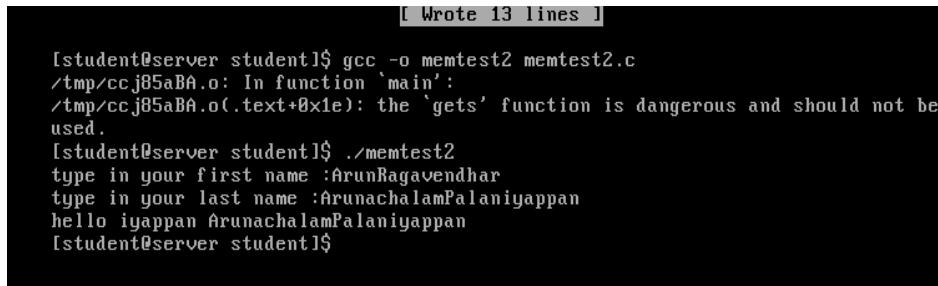
Distance from buf to i = 8 bytes.

- Entering 8 or more characters changed and over wrote i. E.g. - 16+ characters causing i to be over written.



```
root@kali-2019: ~  
File Edit View Search Terminal Help  
| i position | buf start | buf end | i value  
abcdefg | 2965173292 | 2965173284 | 2965173292 | 0  
OK. | 2965173292 | 2965173284 | 2965173292 | 1818978921  
abcdefg | 2965173292 | 2965173284 | 2965173292 | 1818978922  
OK. | 2965173292 | 2965173284 | 2965173292 | 1818977897  
^C  
root@kali-2019:~#  
root@kali-2019:~# ./memtest1  
type in 5-20 characters in the text buffer  
watch the value of i  
it will be corrupted when u exceed 8 chars  
type 'q' to exit :  
| i position | buf start | buf end | i value  
abcdef | 2888792764 | 2888792756 | 2888792764 | 0  
OK. | 2888792764 | 2888792756 | 2888792764 | 0  
OK. | 2888792764 | 2888792756 | 2888792764 | 0  
avcdefghijklmnop | 2888792764 | 2888792756 | 2888792764 | 1818978921  
OK. | 2888792764 | 2888792756 | 2888792764 | 1818978921
```

- In memtest2.c, typing 23 characters into last overflowed into first.
 - Example: first = ArunRagavendhar, last = ArunachalamPalaniyappan
 - Output: hello iyappan ArunachalamPalaniyappan



```
[ Wrote 13 lines ]

[student@server student]$ gcc -o memtest2 memtest2.c
/tmp/ccj85aBA.o: In function 'main':
/tmp/ccj85aBA.o(.text+0x1e): the 'gets' function is dangerous and should not be
used.
[student@server student]$ ./memtest2
type in your first name :ArunRagavendhar
type in your last name :ArunachalamPalaniyappan
hello iyappan ArunachalamPalaniyappan
[student@server student]$
```

Discussion and Application of Learnings

Learning 1:

Buffer overflows happen when user input goes beyond the memory allocated to store it. In memtest1.c, this extra input changed the value of `i`. In memtest2.c, long inputs in last spilled into first, eventually crashing the program.

Real-World Application in Cybersecurity:

Hackers use these flaws to modify variables or inject code. If a program doesn't check input size, it becomes a target—especially in low-level systems like firmware or critical software.

Learning 2:

memtest2.exe crashed when tried in Windows XP. This proves our input reached a sensitive memory area, like a return address.

Real-World Application in Cybersecurity:

This mirrors stack smashing attacks. Hackers analyze crash offsets to find where their exploit landed. Today's systems use defenses like stack canaries, ASLR, and DEP to stop this—but many legacy systems still lack protection.

Learning 3:

Using safe input methods like `fgets()` or limiting input size (e.g., `%10s`) prevents buffer overflows by keeping input within bounds.

Real-World Application in Cybersecurity:

Secure coding—especially in C—is essential. Writing safe input logic and using tools like Valgrind or AddressSanitizer helps catch bugs before attackers do.

Limitations

The lab used unsafe functions like `scanf("%s")` and `gets()` that allow unchecked input, leading to buffer overflows.

In *memtest2.c*, overflowing the last buffer corrupted the first variable—highlighting how adjacent memory can be unintentionally overwritten.

The examples focused on legacy vulnerabilities without covering advanced exploitation or mitigation techniques.