CS253 Assignment

Use After Free

CWE-416

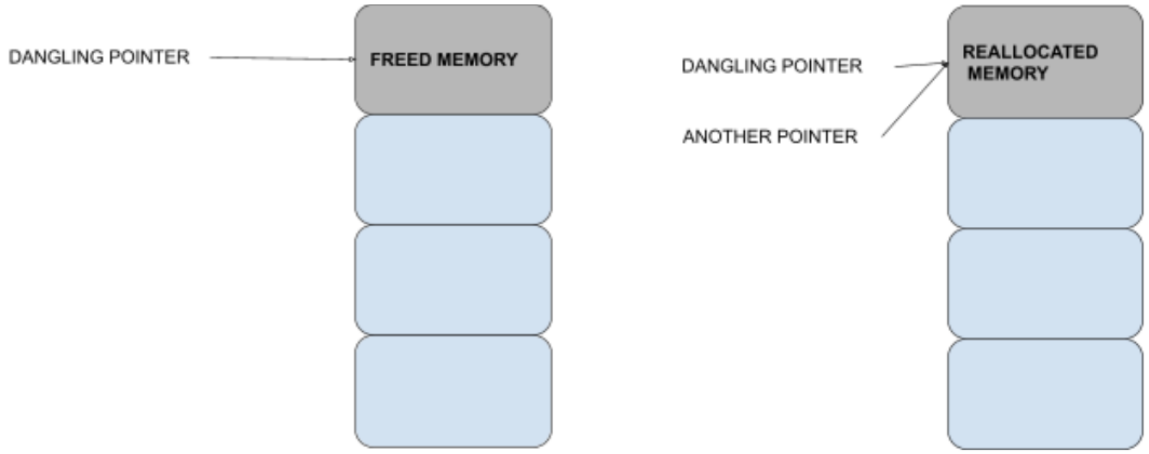
INTRODUCTION

The use after free error occurs when the heap allocated memory that has been freed but the pointer pointing to the freed memory (dangling pointer) is still pointing to the freed memory and is used later in program execution.

Once *free()* is called on an allocated memory , the memory allocator is free to re-allocate that chunk of memory in future calls to *malloc,* although this step is to optimize memory utilization , but become very much vulnerable to attacks.

WHAT IS DANGLING POINTERS?

Dangling pointers are pointers that don't point to a valid memory or object. Dangling pointer arises when destroying or deleting the object/memory pointed by that pointer.



The system may reallocate the previously freed memory pointed by the dangling pointer and if the pointer still dereferences the dangling pointer **it can cause some unpredictable behaviour as the memory may now contain different data or may even contain malicious data injected by an attacker**.

ATTACK MODEL USED

Shell Injection Attack /OS command injection : It is an attack where an attacker exploits some vulnerability to be able to run his own malicious command directly on the server that is running on that application.

LANGUAGE REQUIRED **C,C++**

FUNCTIONALITY OF VULNERABLE C CODE - prog.c

prog.c (vulnerable code) is an interactive c program, which asks the user to select one of the two questions, once the user selects a question , user can display that question using print_question and then answer that question.

Commands used in prog.c program

- *1 :* Allocate a buffer to question 1 pointed by the pointer P1.
  and store the question 1 prepended by echo since it is printed using system command .
- *2 :* Allocate a buffer to question 2 pointed by the pointer P1.
  and store the question 2 prepended by echo since it is printed using system command .
- *Print_question :* pass pointer p1 to print() function which prints the question string since string is prepended by echo.
  ```
  void print(char *s){
        system(s);
  }
  ```
- *Reset_question :* Free the memory allocated to question pointed by P1.
- *Reset_answer* : Free the memory allocated to question pointed by P2.
- *name {any word say X}* : Allocate a buffer to save answer pointed by P2 and store string X in it .
- *END* : End the program execution

How to run vulnerable system to expose the exploit.

There are two ways:

- One is automated in which we need to run “make run1” command which will compile the prog.c and then run the script run.sh which will then run executable piping in the test case from test_suite.
  This will run system command (shell injection) .

  >make run1 //for test_suit
  Or
  >make run3 //for test_suit2

- Other way execute is manually through interacting in terminal using following commands

test_suit	test_suit2
> make vulnerable > .vul > 1 >print_question >reset_question >name ls ; ps ; echo “you are hacked!!” >print_question >END	> make vulnerable >name arpit >reset_answer >1 >name ps ; echo "hacked again!!" >print_question >END

Explain how exploit works

Take the test case commands as (test_suite)

> 1

Allocate a memory in heap of size = SIZE_BUF , pointed by pointer P1.
And stores the string "echo What is your first name , Write in the format {name arpit}".

>print_question

Which call the print(P1) function which call system().

>reset_question

It will free the memory buffer for question , making P1 as a dangling pointer.

>name ls ; ps ; echo “you are hacked!!”

It will assign memory of size SIZE_BUF to save the name , since previously freed memory is also of size SIZE_BUF OS will allocate the previously freed memory to it.

>print_question

Since Pointer P1 which is suppose to be pointed to the Question string now points to the user's Input string.
Thus user can input malicious OS commands as input for name,
which will be executed in print() function since it is passed in system command.

>END

Example 2: (test_suite2)

>name arpit

It will assign memory of size SIZE_BUF to save the name “arpit” , pointed by P2.

>reset_answer

It will free the memory pointed by P2, but P2 still points to that memory as a dangling pointer.

>1

It will assign memory of size SIZE_BUF to save Question string, since previously freed memory is also of size SIZE_BUF OS will allocate the previously freed memory to it.

>name ps ; echo "hacked again!!"

Since P2 both P1 are both pointing to the same memory “ps ; echo "hacked again!!"” is overwritten on the memory which was used by question string.

>print_question

Since pointer P1 is pointing on memory containing “ps ; echo "hacked again!!"” , it will be run as OS Command by system command in print() function.

>END

Note: Prog.c is also vulnerable to double free error , by doing reset_question (or print_question) twice .

Explain how to mitigate this weakness.

This vulnerability can be easily mitigated by Nulling the pointer pointing to the memory after freeing the memory. Since the dangling pointer will not point to any memory now , it cannot be used further for exploitation. This will also mitigate the double free vulnerability.

Not mitigated	mitigated
if(p1!=NULL){ reset(p1); }	if(p1!=NULL){ reset(p1); *p1=NULL; // nulling p1 after freeing.* }

To run prog_mitigated.c on test_suite

> make run2

>make run4