**Secure Systems Engineering (CS6570)**

**Assignment-4**

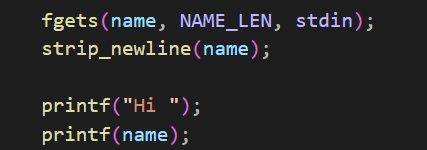
**NAME**: Mohammed Umair Pandit

**ROLL NO:** CS23M037

**Challenge 1:**

The following vulnerabilities/bugs were used to carry out the exploit to retrieve the flag

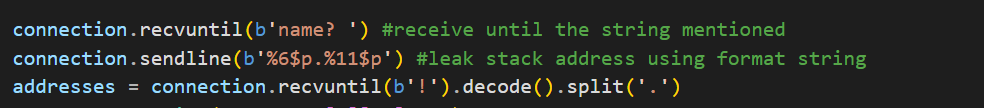
1. **Format String Vulnerability:** A format string vulnerability is a type of security flaw that occurs when a program uses user given data as a format string for functions like printf without proper validation. This can lead to attacks such as Reading from arbitrary memory locations, writing to arbitrary memory locations and executing arbitrary code.



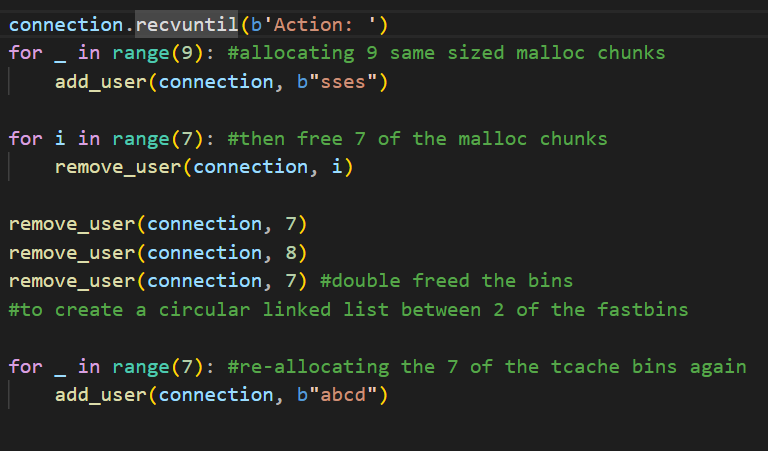
Using this vulnerability, we can leak stack contents such as return addresses, sensitive data etc.

2. **Double free exploit (Heap Exploit)**: refers to a class of memory corruption vulnerabilities that occur when an application frees the same memory allocation twice. This can lead to unpredictable behavior.

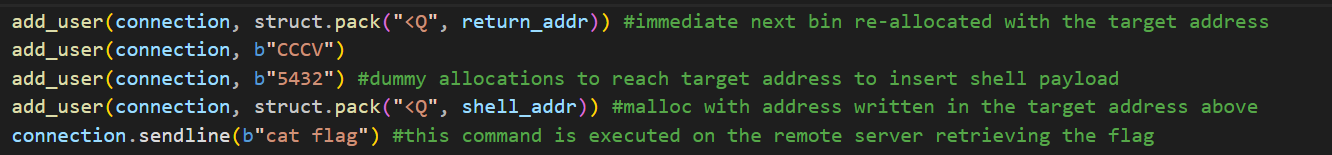
Glibc version used for compiling the binary is 2.27, which is prone to fastbins double free exploit, this is the vulnerability we exploit to access a remote shell and retrieve the flag in the remote server.

In the above code segment, we script when connected to the remote server to receive until we encounter the string given, then we specify a string to send which in format vulnerability of printf function in libc leaks the address of the stack content, which is a major step towards finding possible exploitable addresses.

In this above case we find the return address of the main function where we will use the return address of main to point to a shell accessing the remoter server and cat the flag in the server.

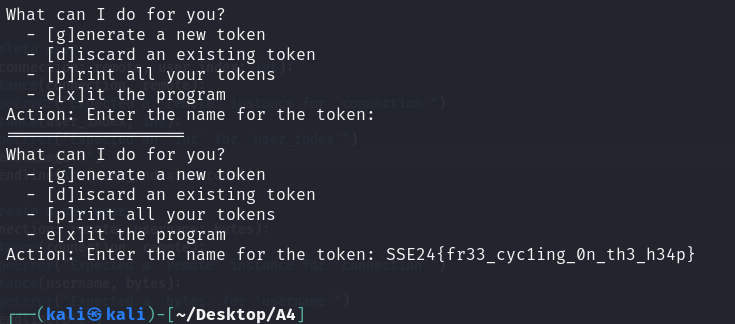


The above code demonstrates how we are using double free vulnerability to move the shell code into the target address which is the return address of the main function. What we have carried out is we know that 7 tcache bins exist , we have allocated 9 same sized malloc chunks , then freed 7 of them and for the remaining 2 double freed one so that there is a circular linked formed in the fastbins of the these 2 chunks which are freed. After we have created the circular structure of these fastbins , we then re allocate the 7 of the tcache bins.



After reallocating the 7 tcache bins, for the last 2 bins next re-allocated bin should contain where the address (target) to be overwritten with. Then we allocate 2 dummy malloc after which next malloc should contain the payload to be written with in the above-mentioned target address.

**Captured Flag snippet:**

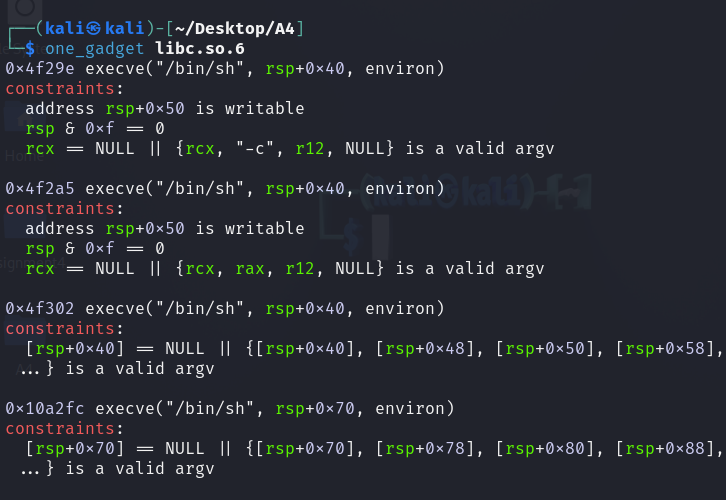
****

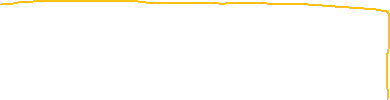


**Challenge 2:**

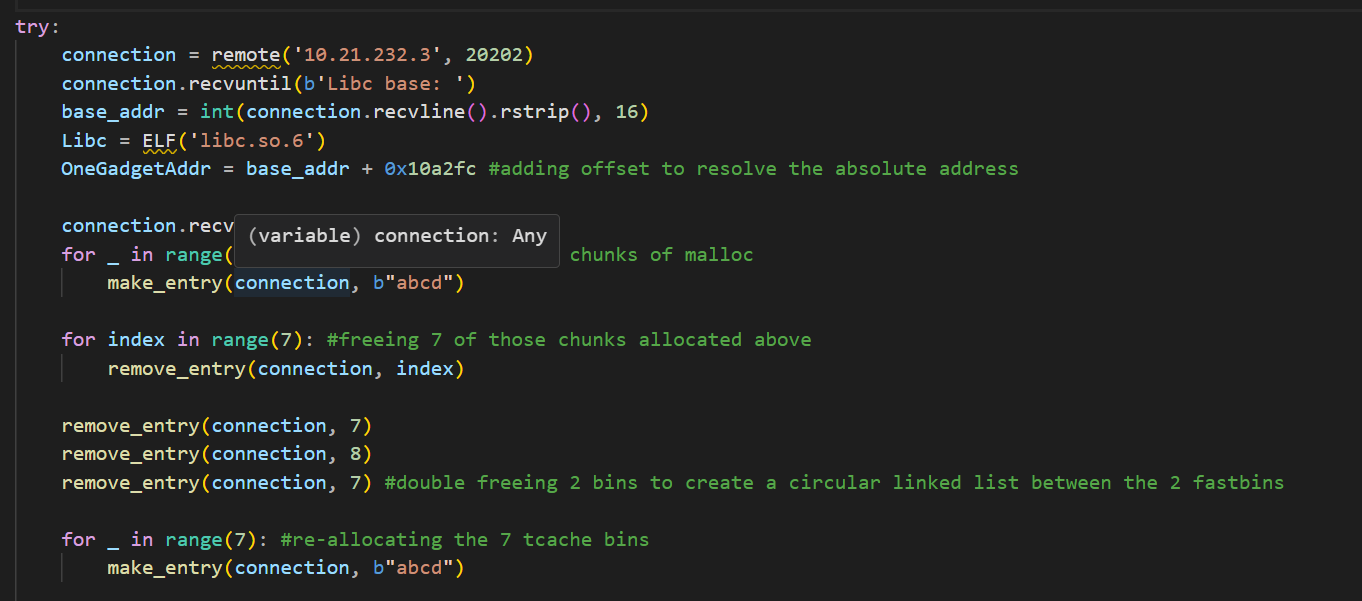
The following vulnerabilities/bugs were used to exploit the remote server and retrieve the flag.

1. **\_\_malloc\_hook:** is a function pointer in the GNU C Library (glibc) that allows developers to intercept and customize memory allocation behavior. It's part of glibc's memory allocation mechanism and allows developers to define their own functions to be called whenever memory allocation functions like malloc() are invoked, which is generally used for dynamically allocating memory in C and C++ programs.. This hook provides a way for developers to implement custom memory management strategies, debugging tools, or perform specific actions whenever memory is allocated.
2. **one\_gadget:** while capturing flags we usually need remote code execution, which leads to call *execve(‘/bin/sh’, NULL, NULL),* it uses symbolic execution to find the constraints of gadgets to be successful. Therefore, in this challenge we will use a single gadget for spawning a shell.
3. **Double Free (Heap Exploit):** double free exploit can be used to overwrite the address of where \_\_malloc\_hook address with OneGadget obtained address (add offset to the libc base address to resolve the absolute address).

****

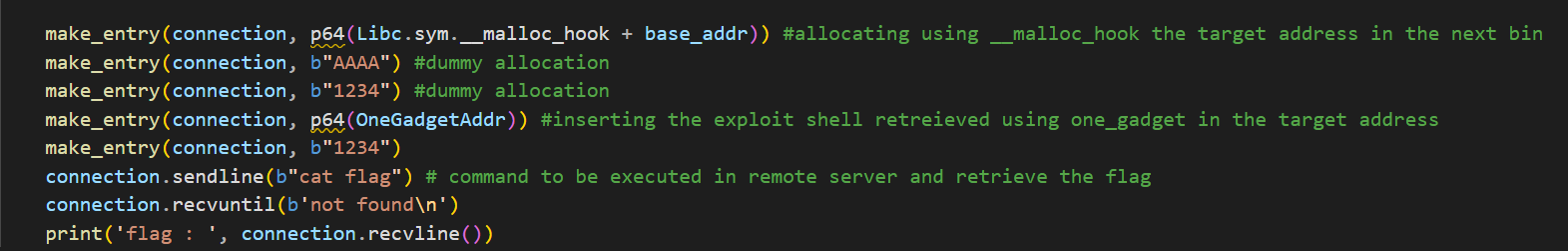


After Installing one\_gadget using Rubygems, using which we can find the gadgets available in a given binary file , lib.so.6 in this scenario , we can find that an call of function as *execve(‘/bin/sh’, \_\_\_, \_\_\_)* is present to spawn a shell using this gadget present in the given binary file.

****In the above snippet of code after starting the connection to the remote server , we receive until a string is encountered as in the given input and a var Libc which is the ELF file of the binary file of the program.

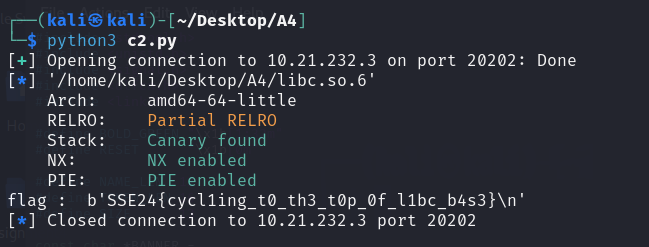
After we initialize one\_gadget address, which is the address of the gadget which we were going to use to redirect the malloc address allocation after using malloc hook in the future. We add a offset to the libc base address to resolve the absolute address so that for each execution as it changes using offset we always reach there to that gadget.

After calculating the gadget address we then as done for previous challenge allocate 9 same sized malloc chunks an then free 7 of those allocated chunks , we then double free a malloc along with a single free of a chunk , to construct a circular linked list between the structure of the the 2 fastbins. We then re-allocate the 7tcache bins present malloc-ing it.

****

Finally, we re-allocate the next bin itself using malloc\_hook the target address in the bin. After 2 dummy malloc allocations, we in 64 bit machine code format insert the exploit shell gadget code in the target address after which we then and there send a string which on the command line cat the file flag to reveal the flag in the flag file , closing the connection.

**Captured Flag Screenshot:**

****

