Homework 2

StudentID: 721370290002

Thursday 28th March, 2024; 21:36

Stack 5

Problem

We call bash to list the functions that are in the program and we find that there is only one function this time.

We try to disassemble the main function to figure out what we are trying to exploit.

There only seems to be a function call `gets` which we can try to maneuver with.

Idea and Attack process

We plan to inject code through the 'gets' call by overflowing the stack such that we can inject some form of code that we want to run.

There are 72bytes to replace from top of stack to ebp. We also have to replace 4 more bytes to remove the ebp of the previous caller then we can inject the address of the shellcode which is 8 bytes after the current ebp. Recall that the block below the ebp is the previous ebp (from push ebp) followed by the return address and following that address will be our shellcode.

We first figure out what is the ebp and esp that we should manipulate with. From the output, we can guess the size of the buffer and we can be sure that the size of the buffer is still 64bytes, and the 76bytes planned initially is valid.

We construct a python file to conjure up the needed values to run our code.

We position our address 50bytes after the ebp such that the address will fall within the nop instructions that is injected. When the CPU runs the nop, it will keep executing till it reaches the shell code that we plan to inject. The shell code was obtained online.

Running the script, we get the shellcode to run and when we prompt ‘whoami‘ now, it returns root instead of user.

Source code

Stack 6

Problem

We try to check what the function actually does and we realise that in `getpath+52` there is some form of comparison with what looks like an address.

Running gdb with the breakpoint at that address will show us that if we try to stack overflow and try to inject another return address such that we run a shell code, the return address changed will be detected. This is because the location that we plan to target will reside above ebp addresses which would have to start with `0xbf000000`. We now have to come up with another way to return the function.

Since there is a check on the address, we want to bypass that check on the address we plan to inject. Recall that the `ret` call copies the address pointed to by esp to eip.

Idea and Attack process

We first find the offset such that we can inject the address to hop back to `ret`. From the error we can work back to find that the offset if 80.

We prepare the script to generate the attack below. We initially started with offsetting at 50 but the script could only run successfully in gdb, and runs into 'Illegal Instruction' when executing in shell. Looking a bit deeper, it could be that the offset is not placing the script with correct alignment in memory. We change the offset to be in multiple of 4s. We also increase the offset to a greater amount so that we avoid the chances of the code being truncated.

Running the code below we see that we gain access into shell as root.

Source code

Stack 7

Problem

Idea and Attack process

We try to find the offset for the target by injecting the string below. Based on the segmentation fault message we can deduce that the offset needed is 80bytes.

We run the code below and we realise that the exploit is not working even after increase the nop sled. We try to find another way perhaps ret2libc.

Lets try to find the address of the lib-c. We use the following commands to obtain the necessary information.

We note that the base address of lib-c is ‘0xb7e97000‘.

We identify the location of the ‘system‘ command in lib-c to invoke it during the attack. The ‘exit‘ command is used so that the attack exits gracefully once we exit the shell. The ‘system‘ command takes in parameters to run, we want to pass in the ‘
bin
sh‘ command to system, one way is to use the address in lib-c that contains it. Through checking lib-c, we find the offset to the string and use it with the base address to obtain the pointer to the string. We should note that the structure of the stack is as follow below which we are trying to emulate. The system call will obtain the parameter 8bytes below ebp, which is past the old ebp and the return address.

With the plan in place, we create the following script and run it.

We run the following script, this time since we want to hold the shell open, we pass in the script as such whereby after the ‘cat‘ command, we pass a ‘-‘ such that the terminal holds the shell open for us to pass in other commands. Reference ‘cat‘ command using ‘man‘ -¿ (cat f - g Output f’s contents, then standard input, then g’s contents.)

Source code