Thursday, July 27, 2023 1:30 PM

Csaw 2018 doubletrouble Pwn 200 (The Floating)

- Doubletrouble has a stack canary, RWX segments, and is a 32-bit binary.
- It takes our inputs and converts them to doubles.
- Proceeding that it does some arithmetic on those doubles, then sorts the doubles least to greatest.
- We get a stack info leak and then confirm it with gdb.
- The info leak is from the stack which starts at '0xfffdd000' and ends at '0xffffe000'.
- But system is at -136824752

Reversing

- When going into main all we need to worry about is 'game()'.
- After the while loop, it prints the address of 'ptrArray' for the infoleak which we see later that is where our input is stored as a double.
- Scans integer into 'heapQt' to make sure it isn't bigger than 64 due to 'ptrArray' can only hold 64 doubles.
- If the int is bigger than 64 than the program will exit and give us the address of the system to taunt us.
- But if it proceeds it enters a for loop which runs 'heapQt' times.
- Each time it goes through the for loop it scans in 100 bytes of data into the heap, then converts it to a double, then stores it into 'ptrArray'.
- Proceeding that, it runs a number of sub functions with 'heapQt' and 'ptrArray' as arguments.
- There is a line in the 'findArray' function which is "*heapQt = v5;" (*param_1 = iVar1;).
- It will dereference 'heapQt' and writes a value to it. This will let us change the value of 'heapQt', which can be passed as an argument to 'sortArray'.
- undefined4 param_3,undefined4 param_4 = double a3, double a4
- A3 = -10 a4 = -100
- A value between -10 and -100 will trigger the write (author uses -23)
- The write appears to increase the value of 'heapQt'.
- 'sortArray' function will loop through the first 'heapQt' doubles of 'ptrArray'.
- It will then compare the value of that double, with the double after it.
- Essentially it just organizes heapQt doubles, starting at the start of ptrArray from smallest to biggest double.

Exploitation

- The bug we now know is that we can overwrite the number of doubles which is stored in 'sortArray'.
- As for that we also have a stack infoleak, an executable leak, and the ability to write data to the stack.
- We will write a greater value to 'heapQt' than 64, that way it will start sorting data past 'ptrArray'.
- Specifically, we will get it to place an address that we want where the return address is stored at ebp+0x4, which will give us code execution.
- But we need to make sure the sorting algorithm leaves the stack canary in the same place or it will make the binary crash.
- To get the return address (0x8049841) overwritten, we can make the value of 'heapQt' that which it extends to two doubles past the return address, which will be the value 69 (hex 0x45).
- To get it to that value we need to play with the amount of inputs we send before/after -23 (which triggers the write) influences it.
- After that we will include three floats which their hex value begins with 0x804.

- ROP gadget --binary doubletrouble | grep 804900a
  0x0804900a : ret
- Since we are limited on what we can call to certain sections of the code, this rop gadget works for us in two reasons.
- First, it is less than '0x8049841' when converted to a float.
- Second, all it does is that it just returns.
- When we go ahead and run the exploit for real, we can just brute force the canary value we need by running the exploit again and again until we get a stack canary value within the range we need.
- The last thing we need to worry about is our shellcode, since we will need to know where it is on the stack to execute it, and we also need to make sure it stays intact and in the correct order after it is sorted.
- We can accomplish this is by appending the 0x90 byte a certain amount of times to the front of certain parts of shellcode.

Quick overview of the challenge
* Program scans in up to 64 doubles, and sorts them from smallest to largest
* Bug in `findArray` allows us to overwrite the float count with a larger value, thus when it sorts the doubles, it will sort values past our input, allowing us to move the return address.
* Format payload to call rop gadget, then shellcode on the stack using stack infoleak. The canary has to be within a set range.
* Format the shellcode to be together after the sorting
* Brute force the stack canary until it is within a range that wouldn't crash our exploit

We have to run the exploit few times for the stack canary to be in the right range but when it works we get the flag.

Flag = flag{4_d0ub1e_d0ub1e_3ntr3ndr3}