Timisoara CTF

Eligible High School Team Scoreboard

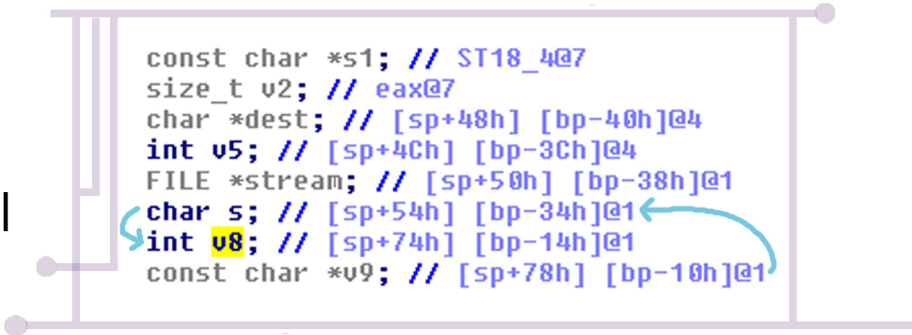
#	Team	Country	Points
1	WreckItRalph		3,581
2	NextLine		3,296
3	dcua		2,131
4	k3rn3l p4n1c		2,086

[+] Attendance (solution: solve_attendance.py)

This was a really easy challenge, by looking closer with a disassembler we can find a special key for triggering a buffer overflow which leads to redirecting the code execution to a special function that cals system.

[+] cparty (solution: solve_cparty.py)

This challenge was a classic buffer overflow with a vulnerable strcpy function, meaning:
by overflowing the v9 local we get to overflow s buffer being able to set the v8 local to -1059127554



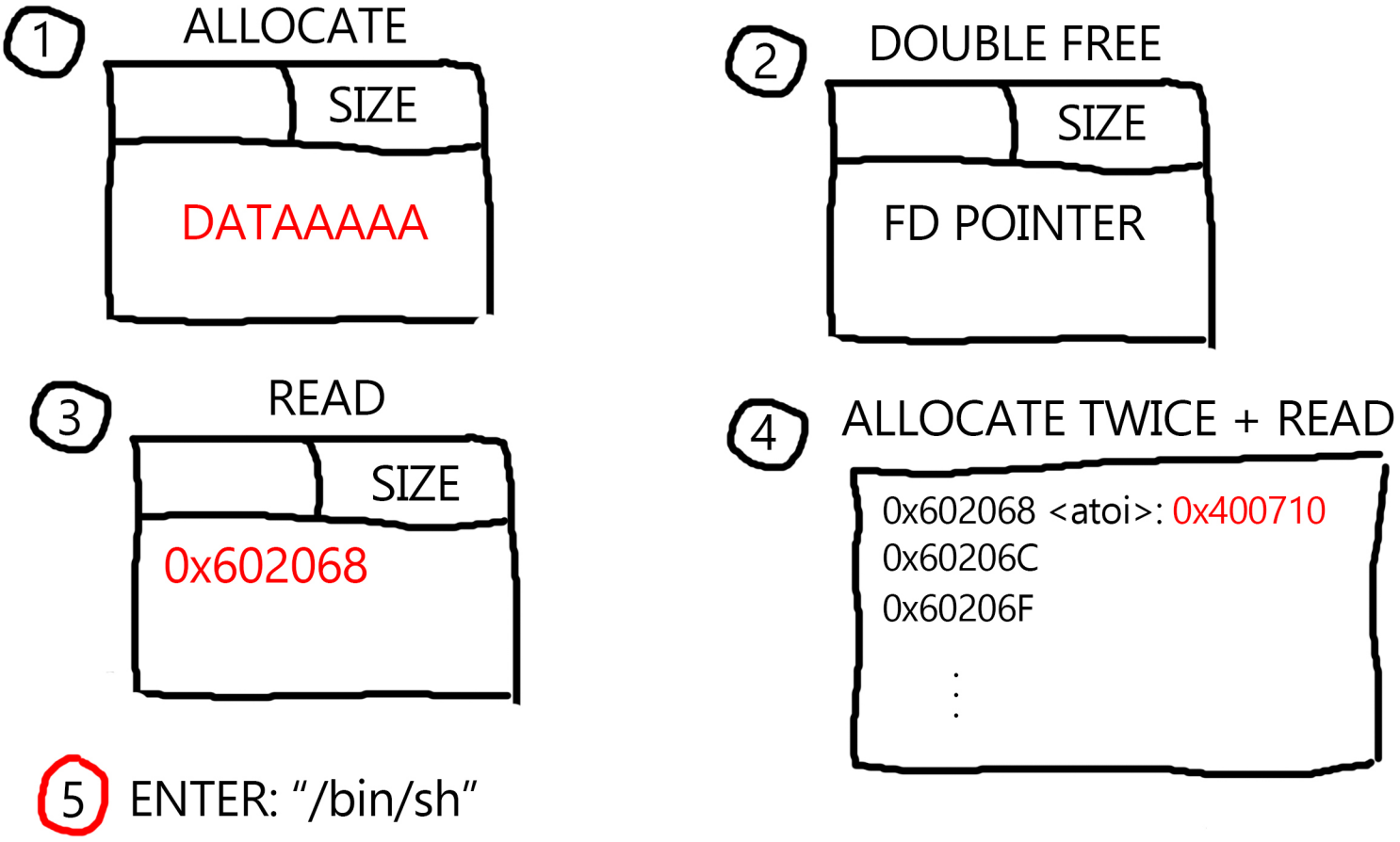
[+] memo (solution: solve_memo.py)

The provided program was opening a file containing the flag and placed it on the stack. After bypassing the POW the program had a vulnerable printf, meaning I could leak the flag using format string vulnerability.

[+] Heap School 101 (solution: solve_heap101.py)

This was a nice challenge, which I enjoyed a lot. Basically it was a classic heap challenge with an interactive menu: Allocate, Free, Write and Read. The vulnerability of this program was UAF, meaning we could still Read or Write to a specific chunk although it was freed. Using this, by performing a double free, a pointer was placed on the heap, pointing to the Forward chunk in the single linked list (because I used chunks under 80bytes). By overwriteing the FD pointer we could allocate wherever in the program memory. So what I did was to allocate once, because fast bins are FIFO, and then allcoate again in the bss segment. Next I used the read action from the menu and overwrite atoi with system. The next step was, instead of entering a menu choice, to enter "/bin/sh" and so I got a shell.

[#] Visual explanation of the attack:



[+] Adrian Pwnescu (solution: solve_adrian.py)

This was a creative challenge. The target was to enter a string with length greater than 10 and no uppercase characters, that matches a randomly generated string. The string was generated using rand() and the seed was printed to screen. Knowing the seed I was able to precalculate the random values, and by doing so, I was able to generate myself the "random string". The vulnerability of the program was that the program was using read, so no null byte was apended at the end of the input. This way I could only enter 10bytes (minimum) and the rest were alraedy in the bufer. The next goal was to match a string starting with no uppercase characters. After running twice the script the flag appeared.