Navdeep Handa (nkh15)

A Tale of 3 Passwords and How After, Like, 4 Days, I Still Type x68 Every Goddamn Time I Google an Assembly Instruction

After spending most of spring break in my room on my laptop in the tropical oasis of Spring City, PA, I've become the master of time, space, and x68. (I'll get it eventually). Slight over-exaggeration aside, join me in the retellings of my journey through the Wild West 🤠 of ELF executables.

The **first executable's** password is the string **PPBNDdCMClBvuIstJjHwwCP**. I found this by simply refusing to look at the massive amount of assembly (generated via objdump -d -Mintel) from 626 MB of executable and instead opting to just run the UNIX strings program on it. When I ran strings on it, I looked through the output to find where the strings relevant to the program were (for example, "Sorry, not correct!"), and saw what I thought was the password. A quick glance at the assembly was good enough for me to confirm no further algorithmic complexity (most of it was library functions with a very small main section in which nothing particularly interesting happened).

The **second executable's** password is the string **3.141593.** I found it through a combination of static analysis of the assembly/symbol table and going through the execution of the program in gdb. Running both the UNIX strings and mystrings generated nothing of interest, so I generated a full disassembly using objdump. I looked at the symbol table of the executable using nm and determined what functions were actually in the source code, and started to construct a call graph (noticing that main() called d(), and d() called more subfunctions, etc.) and debug some of the functions. I quickly realized this was a lot more effort than it was worth. From here, I changed my strategy. I pinpointed where the strings "Sorry, not correct!" and "Congratulations! Unlocked with password %s" were and reverse-engineered backward from there. I examined a lot of registers and debugged a lot of program flow before happening upon the password. I reasoned that the password was probably just a string because my previous work debugging the other functions suggested nothing else in the executable was particularly interesting.

The **third executable** has multiple possible passwords generated by an algorithm. The password must be a minimum of 16 characters for the program to tell you whether you passed or failed (else it will remain in a wait state for further input, as it uses getchar in a loop that runs 16 times to gather input), but the program will ignore everything after the first 16 characters. Exactly 9 (not 8, not 10) of those first 16 must be valid characters, which are characters within the set [c, s, 0, 4, 4, 9] (nice one!). A valid password is **cs0449999idkljfjkli**, and an invalid password is **csdhfddsfdfhgghggh** or **cs0449999idkjjjs.** I found this password the same way as I found the password for the second executable; that is, I tried using UNIX strings, got nothing out of it, and skipped straight to analyzing the disassembly to find, using the output strings to the user, the loop where the password is determined. I did not even attempt analyzing the rest of the assembly, as I learned from debugging the second executable that it is a lot of work to do so, and there was no symbol table or functions that might make that easier.

As a final note, I am reasonably confident that none of the passwords change from day to day because I didn't see any library calls to time functions. I also figured nothing was nondeterministic because I didn't see any calls to random functions. Obviously, I could have fully debugged all the assembly to be sure, but if I'm being honest, I don't care that much.