TD 9: Hackme!

Get the TD tarball from
`https://dept-info.labri.fr/~thibault/SecuLog/td9.tgz`
unpack it. Yes there is only a single binary. It asks a series of passwords that you will have to find out with increasingly complex analysis. I have left some debugging symbols so that navigating in functions etc. is not too complex with gdb. You just do not have the source code available :)

Note : this is of course just an introduction to reverse-engineering. There is ample more to study and train.

Note2 : while you progressively find out passwords, you can copy/paste the whole set in one go to quickly get to the one you are working ont.

Rules

The situation is that you're on a mission, on your own, without Internet access, but you have brought your usual tools with you.

That means you can use your own notes, you can access the course slides.

But you cannot communicate with anybody else or look for something on the Internet.

Level 0

The most basic analysis that you can perform on the binary is the `strings` command. This is enough to pass this level :)

Also use `ltrace` to watch the behavior of the program. It is like `strace`, but at the libraries level, so you can actually see `strcmp` calls.

Level 1

Programs protect their embedded data by ciphering their content, so you cannot easily read it from the file itself with `strings`. But you can easily read it from memory during execution.

You can use `ltrace` to indeed get the cleartext password. But let's see a more technical way to get it, to get more training.

Set a breakpoint on `strcmp@plt` to catch the level 1 attempt to compare your password with the expected password. Note that since `strcmp@plt` has not started at all, `pframe` shows `ret@` etc. not for `strcmp@plt`, but for its caller. So rather go back with `up` to the caller of strcmp (`r1`), and use `disas` to see how the two strings are passed to strcmp, and print them.

(reminder : to print a string from gdb you can use for instance `p (char*) 0x12345678`)

Now that you know where the expected password is in memory, set a watchpoint on its first character to determine the name of the function that achieves the decryption at program startup.

Level 2

Programs rather avoid keeping the unciphered data in the process memory, and rather keep only the ciphered data in memory, and uncipher only when needed. See that `ltrace` is indeed not effective for that level.

Again, you can use a breakpoint on `strcmp@plt` and look at the strings passed to it by its caller `r2`. Notice that `r2` indeed passes the encrypted passwords, not the cleartext passwords. What you thus need to do is find out the encryption algorithm. Compare the code of `r1` used in the previous section and `r2` to see the call that encrypts your input. Disassemble that function to find out the (trivial) encryption algorithm.

Note : you can use `/net/ens/SecuLog/objdump/objdump -d --visualize-jumps hackme` to get a nice view of the jumps, that makes identifying loops very easy. (It also nicely shows the trap cases !)

Note2 : to make things simpler, I made sure that the ciphered password is encoded in ASCII so you can easily copy/paste it. If you get something *really* odd, non-ASCII, you're not looking at the right ciphered password.

Level 3

Using `strcmp` is too obvious to track down, so programs would rather perform the comparison themselves and not call it, so the breakpoint on `strcmp` won't work this time. You can however press control-C while it is asking the password, and use `bt` and `frame` to look at the interesting caller and check what it is doing to compare the password. You can also use `b * 0x12345` to set a breakpoint at a specific instruction.

Level 4

Trivial comparison is a bit too obvious (and values leak out by 4-byte pieces). Continue setting execution with gdb to find out the (not actually complex) ciphering algorithm used by level 4. Explain how the assembly code of the comparison function works.

Level 5

Reversable ciphering is usually weak, level 5 uses something different, explain it and how to pass it. Is this really strong ? How to trivially strengthen it a bit ?