Introduction to Software Reverse Engineering, SU2016

Report: Unveiling L5

HeeHoon Kim

Checking the file

With ‘file’ utility, one can know that L5 is 32-bit x86 executable. If L5 is executed, it prompts the password. However, the password is not revealed with naive way. (e.g. strings, strace, ...) Therefore, I tried to attack with static/dynamic analysis.

Static analysis

I disassembled the file first. The symbols were removed and there was no main function. I know, however, usual C programs start with __libc_start_main, and the argument passed to the function, 0x8048780, could be the address of main function. Starting from 0x8048780, there are codes which do the followings:

1. Function prolog
2. If argc is not 2, it jumps to 0x8048847, prints usage, and exits. If not, it continues.
3. If length of the second argument, i.e. input password, is not 12, it prints error and exits. If not, it jumps to 0x80487d9.
4. By calling valloc, it allocates 0x400 bytes of memory. Let’s call this memory as M.
5. By calling mprotect, it sets M to be read, written, and executed.
6. It loads 0x4d bytes from 0x8048ae0 (from .rodata section), xor each byte with 0xb3, and writes into M.
7. Call M with the pointer to input password as argument. In other words, data in M are regarded as machine codes and executed.

For further analysis, we need to analyze what codes in M do. I extracted 0x4d bytes from .rodata section, xor-ed, and disassembled with objdump. The resultant assembly does the followings:

1. It writes 12 bytes (0xc3f8e1de, 0xd2c0fce2, and 0xfbd1fda2) to stack.
2. Each byte is xor-ed with the corresponding password byte and 0x93.
3. Resultant 12 bytes are or-ed and returned.

After the function, I expected the password check routine but the program just saves the return value to 0x804a050 and returns soon. To track further execution, I performed dynamic analysis.

Dynamic analysis

- Bypassing debugger detection

When I ran the program on gdb, it got SIGKILL. I traced the code, and found that the program was checking the existence of debugger using ptrace. If return value of ptrace is -1, it means that the program is already ptraced (usually by gdb), so it halts. I set a breakpoint at the check code (near 0x804875b), and set %eax register to 0 each time after ptrace returned, effectively preventing the program from halting.

- Finding password check code

I stepped the code more after M is called, and found the check code after exit() is called. (I guess the check code is on the exit handler.) It checks the value at 0x804a050, and shows success message if the value is 0.

Conclusion

The correct password is dee1f8c3e2fcc0d2a2fdd1fb XORed with 939393…, which is MrkPqoSA1nBh in ascii text.