

# oguzbey Lucky Numbers

<https://crackmes.one/crackme/5e567e1d33c5d4439bb2dca0>

Crackme writeup by [@H0I3BI4ck](#)

<https://twitter.com/H0I3BI4ck>

crackmes.one user [b1h0](#) <https://crackmes.one/user/b1h0>

Date: 18/abr/2020

You can download **lucky\_numbers** from this [link](#).

We have here an executable in **ELF** format (Linux).

Import Results Summary	
	
Project File Name:	lucky_numbers
Last Modified:	Fri Apr 17 19:29:43 CEST 2020
Readonly:	false
Program Name:	lucky_numbers
Language ID:	x86:LE:32:default (2.9)
Compiler ID:	gcc
Processor:	x86
Endian:	Little
Address Size:	32
Minimum Address:	08048000
Maximum Address:	_elfSectionHeaders::000000c7
# of Bytes:	557
# of Memory Blocks:	6
# of Instructions:	0
# of Defined Data:	7
# of Functions:	1
# of Symbols:	1
# of Data Types:	13
# of Data Type Categories:	2
Created With Ghidra Version:	9.1.1
Date Created:	Fri Apr 17 19:29:43 CEST 2020
ELF File Type:	executable
ELF Original Image Base:	0x8048000
ELF Prelinked:	false
Executable Format:	Executable and Linking Format (ELF)
Executable Location:	oguzbey-Lucky Numbers/lucky_numbers
Executable MD5:	007e70da5226757d3900b04aefc5c09e
Executable SHA256:	6dc056d56f71c9e75a5470721bcaa28806066d8a47fb7658c4d718eb71149e0f
FSRL:	file:///oguzbey-Lucky Numbers/lucky_numbers?MD5=007e70da5226757d3900b04aefc5c09e
Relocatable:	false

Shows us only the message **"Lucky Numbers"** and expects us to enter, apparently a number.

```
Parrot Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
[b1h0@parrot]-[/mnt/programaciovmrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
└─$ ./lucky_numbers
Lucky Numbers: 66
Sorry :((
[x]-[b1h0@parrot]-[/mnt/programaciovmrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
└─$
[x]-[b1h0@parrot]-[/mnt/programaciovmrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
└─$ ./lucky_numbers
Lucky Numbers: 7777
Sorry :((
[x]-[b1h0@parrot]-[/mnt/programaciovmrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
└─$ 77
bash: 77: orden no encontrada
[x]-[b1h0@parrot]-[/mnt/programaciovmrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
└─$
```

But something very curious happens. If the number has more than two digits, the remaining digits are tried to be executed in console after the program ends.

So we already have a first clue, and that is that the number has to be a value between 0 and 99.

And we have a side effect that could lead to the execution of a command.

Take note of the following screenshot.

```
Parrot Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
[b1h0@parrot]-[/mnt/programaciovmrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
└─$ ./lucky_numbers
Lucky Numbers: 99ls
Sorry :((
[x]-[b1h0@parrot]-[/mnt/programaciovmrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
└─$ ls
'blh0-oguzbey-Lucky Numbers.md'  img  lucky_numbers
[x]-[b1h0@parrot]-[/mnt/programaciovmrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
└─$
```

Let's do the static analysis ...

## Ghidra's static analysis

We start by analyzing the data section and find the following texts. Assigning labels for better identification.

The screenshot shows a debugger window titled 'Listing: lucky\_numbers' with a sub-tab 'lucky\_numbers'. It displays a memory dump of the 'lucky\_numbers' section. The dump is organized into three blocks: 'DAT\_MSG\_LUCKY\_NUMBERS', 'DAT\_MSG\_GOOD\_JOB', and 'DAT\_MSG\_SORRY'. Each block contains a list of memory addresses, hex values, and their corresponding ASCII characters. Cross-references (XREF) are shown on the right side of each block.

Address	Hex	ASCII
0804a000	4c	??
0804a001	75	??
0804a002	63	??
0804a003	6b	??
0804a004	79	??
0804a005	20	??
0804a006	4e	??
0804a007	75	??
0804a008	6d	??
0804a009	62	??
0804a00a	65	??
0804a00b	72	??
0804a00c	73	??
0804a00d	3a	??
0804a00e	20	??

  

Address	Hex	ASCII
0804a00f	47	??
0804a010	6f	??
0804a011	6f	??
0804a012	64	??
0804a013	20	??
0804a014	4a	??
0804a015	6f	??
0804a016	62	??
0804a017	20	??
0804a018	21	??
0804a019	0a	??

  

Address	Hex	ASCII
0804a01a	53	??
0804a01b	6f	??
0804a01c	72	??
0804a01d	72	??
0804a01e	79	??
0804a01f	20	??
0804a020	3a	??
0804a021	28	??
0804a022	28	??

Here we have the code part where the success or error messages are shown, where it is verified that it makes calls to the **Linux Syscalls**.

You can find a reference to these calls at the following link: <https://syscalls.kernelgrok.com/>

The screenshot shows a debugger window titled 'Listing: lucky\_numbers' with a sub-tab 'lucky\_numbers'. It displays assembly code for the 'SHOW\_MSG\_SORRY' and 'SHOW\_MSG\_GOOD\_JOB' functions. The code includes instructions like MOV, INT, and CALL, along with comments explaining the operations. Cross-references (XREF) are shown on the right side of each block.

```

//
// .text
// SHI_PROGBITS [0x8049000 - 0x804908c]
// ram: 08049000-0804908c
//

SHOW_MSG_SORRY
XREF[4]: 0804805c(*), 0804907e(j), 08049083(j),
         _elfSectionHeaders::00000034(*)

08049000 b8 04 00 00 00 MOV EAX,0x4          system call sys_write
08049005 bb 01 00 00 00 MOV EBX,0x1          fd
0804900a b9 1a a0 04 08 MOV ECX,DAT_MSG_SORRY  pointer to string "Sorry"
0804900f ba 0a 00 00 00 MOV EDI,0xa        number of chars to print
08049014 cd 80          INT 0x80          call sys_write
08049016 b8 01 00 00 00 MOV EAX,0x1          system call sys_exit
0804901b cd 80          INT 0x80          call sys_exit

SHOW_MSG_GOOD_JOB
XREF[1]: 0804908b(j)
         system call sys_write
0804901d b8 04 00 00 00 MOV EAX,0x4          fd
08049022 bb 01 00 00 00 MOV EBX,0x1          pointer to string "Good Job"
08049027 b9 0f a0 04 08 MOV ECX,DAT_MSG_GOOD_JOB number of chars to print
0804902c ba 0b 00 00 00 MOV EDI,0xb        call sys_write
08049031 cd 80          INT 0x80          system call sys_exit
08049033 b8 01 00 00 00 MOV EAX,0x1          call sys_exit
08049038 cd 80          INT 0x80

```

Next, we have the main function, which begins at the entry point, and is the one that we have to analyze more deeply to understand how the number entered by the user is verified.

```

Listing: lucky_numbers
lucky_numbers X
*****
*                               *
*                               *
*****
FUNCTION
*****
undefined __cdecl entry(void)
AL:1 <RETURN>
entry
XREF[2]: Entry Point(*), 08048018(*)
0804903a b8 04 00 00 00 MOV EAX,0x4      system call sys_write
0804903f bb 01 00 00 00 MOV EBX,0x1      file descriptor (fd)
08049044 b9 00 a0 04 08 MOV ECX,DAT_MSG_LUCKY_NUMBERS pointer to string
08049049 ba 0f 00 00 00 MOV EDI,0xf      number of chars to print
0804904e cd 80      INT 0x80      call sys_write
08049050 b8 03 00 00 00 MOV EAX,0x3      system call sys_read
08049055 bb 02 00 00 00 MOV EBX,0x2      fd
0804905a b9 24 a0 04 08 MOV ECX,DAT_INPUT_NUMBER input char buffer
0804905f ba 02 00 00 00 MOV EDI,0x2      char count to input
08049064 cd 80      INT 0x80      call sys_read
08049066 a0 24 a0 04 08 MOV AL,[DAT_INPUT_NUMBER] load first character to AL
0804906b 2c 30      SUB AL,0x30      subtract 48 dec, which is the ASCII
                                value of 0, to get the value.
0804906d 8a 1d 25 a0 04 08 MOV BL,byte ptr [DAT_INPUT_2_NUMBER] load second char to BL
08049073 80 eb 30      SUB BL,0x30      subtract 48 dec (char '0'),
                                to convert to numeric value.
08049076 10 d8      ADC AL,BL      add the AL + BL values with carry.
08049078 27      DAA      adjust the sum of packed values to be valid BCD
08049079 80 c3 30      ADD BL,0x30
0804907c 3c 16      CMP AL,0x16      Compare with 0x16 : BCD 0001 0110 = 1 6
                                AL is the sum of first and second digit in BCD.
                                So, first digit = 16 - second digit.
0804907e 75 80      JNZ SHOW_MSG_SORRY
08049080 80 fb 38      CMP BL,0x38      Compare with 0x38 : BCD 0011 1000 = 3 8
                                BL must be firsts digit plus 0x30
                                So the second digit has to be an "8".
08049083 0f 85 77 ff ff ff JNZ SHOW_MSG_SORRY
08049089 39 c0      CMP EAX,EAX
0804908b 74 90      JZ SHOW_MSG_GOOD_JOB

```

After entering the numerical value, which we already know **must be 2 digits**, it makes the corresponding conversion of each digit to its numerical value subtracting the **ASCII value 0x30** (character code "0").

Then we can see that it adds the first digit and the second and leaves its value in the **AL** register, subsequently making a correction with **DAA** so that the value that remains in the register is the result of the sum in **BCD format**.

Since the comparison with AL is with **0x16**, it means that the sum of the two digits must be **16 dec**, so the valid combinations would be 79, 97 or 88.

Then we see that the result of the second digit adds 0x30 again and compares with the value **0x38**, therefore it indicates that the second digit must be an **"8"**.

So the crackme solution is **88**.

Let's see if I'm right ...

```

Parrot Terminal
Archivo  Editor  Ver  Buscar  Terminal  Ayuda
[b1h0@parrot]-[/mnt/programaciomvrev/CrackMe/crackmes.one/oguzbey-Lucky Numbers]
$ ./lucky_numbers
Lucky Numbers: 88
Good Job !

```

## That's all folks!

