Solution for
lord's easy linux crackme

pNg

May 21, 2005

Contents

1 General info about the crackme

- It's an i386 Linux executable written in assembly language.
- It will print a message under certain circumstances.

According to the description, thr crackme will print text under certain circumstances, and it's our task to find and possibly circumvent them by changing the program's execution flow.

2 Rules for solving the crackme

No special rules for solving this crackme have been defined, so basically we can do everything we want ;-)

3 Tools used for solving the crackme

I used the following tools to solve this crackme:

- `objdump` from the GNU binutils package
- `gdb`, the GNU source level debugger

4 Actually solving the crackme

In this section, I'll explain the steps that where necessary to solve this crackme.

4.1 Step 1: Analyzing it

To get some general overview, I first tried to run the crackme without any parameters and had a look at its return code to the OS:

```
png@silver:~/crackmes/blah$ ./blah
png@silver:~/crackmes/blah$ echo $?
0
png@silver:~/crackmes/blah$
```

As i expected, the crackme prints nothing and exits with a normal return code, so the "certain circumstances" that were mentioned in the description are not met.

4.2 Step 2: Disassembling the binary

Now I disassembled the crackme's binary with `objdump` from the gnu binutils package which gave me the following assembly listing:

```
blah.orig:     file format elf32-i386

Disassembly of section .text:

08048094 <.text>:
 8048094: 31 c0                 xor    %eax,%eax
 8048096: b8 2f 00 00 00        mov    $0x2f,%eax
 804809b: cd 80                 int    $0x80
 804809d: 3d ad de 00 00        cmp    $0xdead,%eax
 80480a2: 75 16                 jne    0x80480ba
 80480a4: b8 04 00 00 00        mov    $0x4,%eax
 80480a9: bb 01 00 00 00        mov    $0x1,%ebx
 80480ae: b9 c4 90 04 08        mov    $0x80490c4,%ecx
 80480b3: ba 06 00 00 00        mov    $0x6,%edx
 80480b8: cd 80                 int    $0x80
 80480ba: 31 c0                 xor    %eax,%eax
 80480bc: 40                    inc    %eax
 80480bd: 31 db                 xor    %ebx,%ebx
 80480bf: cd 80                 int    $0x80
```

4.3 Step 3: Analyzing the disassembly listing

Next step is to closely examine the assembly source provided by `objdump` to get more knowledge about what the program exactly does:

Offset	Instruction	Description
0x8048094	`xor %eax, %eax`	Clear out EAX
0x8048096	`mov $0x2f, %eax`	EAX=47 (*getgid*)
0x804809b	`int $0x80`	System call interrupt

In the first three lines, the crackme issues Linux system call number `47`, which is *getgid* according to the Linux syscall index. The syscall's return value, which is the group id if the user that runs the crackme, is stored in EAX after return from the syscall.

Offset	Instruction	Description
0x804809d	`cmp $0xdead, %eax`	Compare EAX with `0xDEAD`
0x80480a2	`jne 0x80480ba`	Jump to `0x80480ba` if values where not equal

Here, the return value from the above syscall is compared to `0xDEAD` (57005 decimal) and if they're not equal, the program will jump to `0x80480ba`:

Offset	Instruction	Description
0x80480ba	`xor %eax, %eax`	Clear out EAX
0x80480bc	`inc %eax`	EAX=1 (*exit*)
0x80480bd	`xor %ebx, %ebx`	Clear out EBX
0x80480bf	`int $0x80`	System call interrupt

At this offset, system call number `1` is used to *exit* the program. This syscall takes the program's return code as an argument in EBX, so the program will exit with return code 0 as we saw earlier.
Now we know that the program will exit without any further action when the user who's running it is not in group 57005, so it should be easy to get the hidden message.

5 Circumventing the crackme's protection

We actually have two ways to circumvent this crackme's protection:

- Create a new group with gid 57005 and add an user to it to run the program.
- Patch the program so that any user can see the hidden message.

The latter one seems more interesting, so lets try it:

5.1 Patching the program

Patching this crackme is very ease because it has only one conditional jump that locks us out from seeing the hidden message:

```
80480a2: 75 16                  jne    0x80480ba
```

If we remove this conditional jump from the program, it will display the message regardless of the user's group id. We can remove the jump by replacing it's opcodes (`0x75 0x16`) with NOPs:

```
80480a2: 90                    nop
80480a3: 90                    nop
```

This can be done by loading the binary with `gdb` and then issuing the following commands:

```
png@silver:~/crackmes/blah$ gdb --write -nx -q blah
(gdb) x/x 0x80480a2
0x80480a2: 0x04b81675
(gdb) set {int} 0x80480a2 = 0x04b89090
(gdb) q
png@silver:~/crackmes/blah$
```

After modifying the binary this way, we should see the message regardless of our gid:

```
png@silver:~/crackmes/blah$ ./blah
Okej!
png@silver:~/crackmes/blah$
```

Bingo, it worked :-)

6 Conclusion

This crackme was one of the easiest you can possibly find on the net, and almost everyone with a little knowledge of assembly language should be able to solve it straight forward.