

Joseph Guzman

12/22/2019

Leviathan01

This is the login info for this lab:

```
ssh server: leviathan.labs.overthewire.org
port: 2223
username: leviathan1
password: rioGegei8m
```

Once you login you'll notice a directory named **check**. It is a 32-bit lsb executable. I know this because I ran the **file** command on it. See *figure 1* for details.

A terminal window screenshot showing the command 'file check' being executed. The output indicates that 'check' is a setuid ELF 32-bit LSB executable for Intel 80386, version 1 (SYSV), dynamically linked, with interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, built for a GNU/Linux environment. The SHA1 hash is also displayed. The terminal prompt is 'leviathan1@leviathan:~\$'. At the bottom, there is a status bar with 'U:**@ *shell*' and 'All (3,24) (Shell:run)'.

Figure 1: file details

When you execute the file it prompts you for a password. We can assume that it gives us the password of the Leviathan2 if we guess the right password. I tried two things which turned out to be overkill and not really work.

The first thing I did was use retdec to decompile the lsbl assembly code. I've included the result code in the github repo as leviathan1.c. This was a logical approach. I did notice in the C code that the user input was being compared to this int: `int32_t v1 = 0x786573; //`
`bp-24` via the strcmp function. When I converted this to ASCII I got the string "xes". I attempted to use "xes" as the password and later found out that the password was "sex". This could have been a bug in retdec or just an inconsistency. It doesn't make sense anyway to use strcmp on between an int and a string.

The second thing I tried was using gdb to edit and read the assembly live. I tracked down exactly where the password check was taking place. It was check with jne instruction. So, I changed the eax register directly, before the check was taking place (see *Figure 2*). This did allow me to bypass the password entry, and get the sh shell prompt. However, for some reason I only got the shell as user leviathan1 instead of leviathan2 as shown in *Figure 3*. [This stackoverflow link](#) claims that this is because you cannot debug a setuid binary properly using gdb. A setuid binary is a binary that when executed will execute with the permissions of the owner of the file.

```
File Edit View Search Terminal Tabs Help
leviathan1@leviathan: ~
joseph@joseph-ThinkPad-E580: ~/overthewire-solutions/leviathan/leviathan01

0x004859c <main+97>  mov    %al, -0xa(%ebp)
0x004859f <main+100>  movb   $0x0, -0x9(%ebp)
0x00485a3 <main+104>  sub    $0x8, %esp
0x00485a6 <main+107>  lea    -0x10(%ebp), %eax
0x00485a9 <main+110>  push   %eax
0x00485aa <main+111>  lea    -0xc(%ebp), %eax
0x00485ad <main+114>  push   %eax
0x00485ae <main+115>  call   0x00483b0 <strcmp@plt>
0x00485b3 <main+120>  add    $0x10, %esp
> 0x00485b6 <main+123>  test   %eax, %eax
0x00485b8 <main+125>  jne    0x00485e5 <main+170>
0x00485ba <main+127>  call   0x00483e0 <geteuid@plt>
0x00485bf <main+132>  mov    %eax, %ebx
0x00485c1 <main+134>  call   0x00483e0 <geteuid@plt>
0x00485c6 <main+139>  sub    $0x8, %esp
0x00485c9 <main+142>  push   %ebx
0x00485ca <main+143>  push   %eax
0x00485cb <main+144>  call   0x0048410 <setreuid@plt>
0x00485d0 <main+149>  add    $0x10, %esp
0x00485d3 <main+152>  sub    $0xc, %esp

native process 16198 In: main
0x00485ad in main ()
0x00485ae in main ()
0x00485b3 in main ()
0x00485b6 in main ()
(gdb) info register eax
eax                0xffffffff      -1
(gdb) set $eax = 0
(gdb) □
```

Figure 2: Manually Bypass Password Check

```
File Edit View Search Terminal Tabs Help
leviathan1@leviathan: ~
joseph@joseph-ThinkPad-E580: ~/overthewire-solutions/leviathan/leviathan01

$ whoami
leviathan1
$ □
```

Figure 3: Continue Program, Enter Shell

After making these attempts I googled a solution online. [This](#) link helped me. Basically all that was required was checking the password once with ltrace which shows you the strcmp comparison. We can clearly see that our password attempt is being compared to the string “sex” (see *Figure 4*). So we know that this was the password. It’s insecure that the password is stored raw within the program. It would be smarter to hash the password as is standard in web applications.

```
leviathan1@leviathan:~$ file check
check: setuid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=c735f6f3a3a94adcad8407cc0fda40496fd765dd, not stripped
leviathan1@leviathan:~$ ls
check
leviathan1@leviathan:~$ ltrace ./check
__libc_start_main(0x804853b, 1, 0xffffdc64, 0x8048610 <unfinished ...>
printf("password: ") = 10
getchar(1, 0, 0x65766f6c, 0x646f6700password: asdf
) = 97
getchar(1, 0, 0x65766f6c, 0x646f6700) = 115
getchar(1, 0, 0x65766f6c, 0x646f6700) = 100
strcmp("asd", "sex") = -1
puts("Wrong password, Good Bye ...Wrong password, Good Bye ...
) = 29
+++ exited (status 0) +++
leviathan1@leviathan:~$ ./check
password: \u@\h:\w$
\u@\h:\w$ ls
check
\u@\h:\w$ whoami
leviathan2
\u@\h:\w$ cat /etc/leviathan_pass/leviathan2
ougahZi8Ta
\u@\h:\w$
```

Figure 4: Ltrace Solution

This is the login info for the next lab:

```
ssh server: leviathan.labs.overthewire.org
port: 2223
username: leviathan1
password: rioGegei8m
```