Work & documentation notes of various the Leviathan wargame

Galen Rowell

August 5, 2020

Leviathan

leviathan0

Password to enter: *leviathan0*
Challenge: Within a hidden folder inside the home directory, there was a *bookmarks.html* file. With a quick visual inspection the password is listed within the file. The file is long, and a more suitable method for anything larger or more complex would be a regex search with **grep**.

```
grep 'password' bookmarks.html
```

leviathan1

Password to enter: *rioGegei8m*
Challenge: This level provides a Linux executable which, with the correct password, launches us into a shell of the next leviathan level. From there we can read the password of *leviathan2*.

The shell command **file** is used to test the encoding & file-type of a given file, which is particularly useful on binaries & executable files. The latter part of **file**'s output *"not stripped"* informs us that the debugging symbols were included in this last compilation. This is particularly useful as it allows us to easily trace the given file.

Various debugging and executable-tracing commands exist, such as **gdb, strace, ltrace & sysdig**. **ltrace** is fantastic tool which aims at tracing the execution of a given executable, with particular focus on library calls. **strace** is comparison similar to **ltrace**, except with a heavier focus upon system calls.

With these two commands, one can see **line #12** shows the password for the executable.

the shell during reversal

```
1  file ./check
2    check: setuid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
3    dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32,
4    BuildID[sha1]=c735f6f3a3a94adcad8407cc0fda40496fd765dd, not stripped
5  ltrace ./check
6    __libc_start_main(0x804853b, 1, 0xffffd774, 0x8048610 <unfinished ...>
7    printf("password: ")                                          = 10
8    getchar(1, 0, 0x65766f6c, 0x646f6700password: testPassword
9    )                          = 116
10   getchar(1, 0, 0x65766f6c, 0x646f6700)                         = 101
11   getchar(1, 0, 0x65766f6c, 0x646f6700)                         = 115
12   strcmp("tes", "sex")                                          = 1
13   puts("Wrong password, Good Bye ..."Wrong password, Good Bye ...
14   )                           = 29
15   +++ exited (status 0) +++
```

Note: *Line #8: "testPassword" was manually entered*
Note: *Line #12: the executable checks our input with the string "sex", the password for the script*

leviathan2

Password to enter: *ougahZi8Ta*
Challenge: ltrace is a fantastic tool for discovering the exploit here, it's use reveals two important function calls.

```
access("filename", 4)
```

```
system("/bin/cat filename"file content
```

Note: *'file content' is output by* ***ltrace*** *as part of the executable examination*

These two functions check for read permissions of the file and pass the filename to the command line receptively. The flaw in the script lies in the quotations, the double quotes around the filename are dropped for the shell call.

This causes issues with a spaced filename, as **cat** will print out each argument given. This can be taken advantage of with the following:

```
echo "file a" > a
echo "file a b" > 'a b'
ln -s /etc/leviathan_pass/leviathan3 b
~/printfile "a b"
```

Note: *Files may need to be given read access to the others group*

leviathan3

Password to enter: *Ahdiemoo1j*
Challenge: This level requires a similar methodology to *leviathan1* to solve, a password is required and is compared to a string within the executable. Interestingly, there are several string comparison functions with *strcmp()*, but with the use of **ltrace -S file.sh** the appropriate function call does not get traced. **ltrace** traces library calls, and with the **-S** flag system calls are traced as well.

```
1  ltrace -S ./level3
2  ...various irrelevant setup system calls...
3
4  __libc_start_main(0x8048618, 1, 0xffffd794, 0x80486d0 <unfinished ...>
5  strcmp("h0no33", "kakaka")                                    = -1
6  printf("Enter the password> " <unfinished ...>
7  SYS_fstat64(2004, 0xffffd080, 0xf7ee8245, 0xf7fc3960)         = 0
8  SYS_brk(0xf7fc5000)                                           = 0x804b000
9  SYS_brk(0xf7fc5000)                                           = 0x806c000
10 <... printf resumed> )                                        = 20
11 fgets( <unfinished ...>
12 SYS_fstat64(2004, 0xffffd3b0, 0xf7ee8245, 0xf7fc3960)         = 0
13 SYS_write(20, "Enter the password> ", 4159605747Enter the password> )
       = 20
14 SYS_read(1024test
15 , "test\n", 4159605635)                          = 5
16 <... fgets resumed> "test\n", 256, 0xf7fc55a0)                = 0xffffd5a0
17 strcmp("test\n", "snlprintf\n")                               = 1
18 puts("bzzzzzzzzap. WRONG" <unfinished ...>
19 SYS_write(19, "bzzzzzzzzap. WRONG\n ", 4159605747bzzzzzzzzap. WRONG
20 )          = 19
21 <... puts resumed> )                                          = 19
22 SYS_exit_group(-134453124 <no return ...>
23 +++ exited (status 0) +++
```

Note: *Line #17 is of the most interest here, the rest is irrelevant*

leviathan4

Password to enter: *vuH0coox6m*
Challenge: This is a simple level which requires converting binary encoding into ASCII text. The easiest solution is to copy-paste the string to an online website for conversion, an alternative is provided below using Perl.

source & explanation

```
echo 01000001 01000010 | perl -lape '$_=pack"(B8)*",@F'
```

leviathan5

Password to enter: *Tith4cokei*
Challenge: There is a *SETUID* executable in the home directory, it reads the file */tmp/file.log*, then deletes it. Inspection using **ltrace -S**, reveals that the executable does not check if the file is a symbolic link.

```
ln -s /etc/leviathan_pass/leviathan6 /tmp/file.log
```

leviathan6

Password to enter: *UgaoFee4li*
Challenge: This level has yet another *SETUID* executable in the home directory. When prompted for use the response is:

```
~$ ./leviathan6
usage: ./leviathan6 <4 digit code>
```

As this requires brute-forcing, we'll create a script to input the 9999 possible passcode combinations:

```
#!/bin/bash
for num in {0000..9999}
do
  echo "$num"
  /home/leviathan6/leviathan6 $num
  sleep 0.001
done
```

Note: *This script will enter the pause when we enter the correct passcode and enter the leviathan7 shell, then resume on exit*

A write-up of assembly code analysis, instead of brute-forcing

leviathan7

Password to enter: *UgaoFee4li*

This level has no challenge, it's the final level of the bandit wargame.

Links & resources

1. When scripting, it is often useful to have a temporary directory where files can be created & modified without the risk of littering such files about the filesystem. So a temporary directory (often in /tmp/) is useful, **mktemp** does this:

   move to the new temporary directory

   ```
   cd $(mktemp -d)
   ```

store the new temporary directory path

```
tmp_dir=$(mktemp -d)
```

2. Git is has many fantastic functionalities, here are some key ones:

compare working tree with commited version

```
git diff <filename>
```

reset working tree file to the commited version

```
git checkout -- <filename>
git restore <filename>
```

Note: *These two methods are destructive, and discard any changes,* ***git stash*** *may be more suitable, as it backs up the working tree/changes*