Leviathan Guide:

Overview:

This is a guide to the Leviathan wargame on overthewire.org. The goal is to document my thought process while working through each level while also being able to possibly guide others who may need a hint or some help on these challenges, just like I had done with the Bandit game. This wargame only has 7 levels, but none of the levels have level goals. These challenges are going to be much more open ended than a game like Bandit, meaning this will have much more thought process documented than the bandit guide because I don't have any direction upon starting a challenge other than the overarching goal of obtaining a password for the next level.

Useful Info:

Just like in Bandit, there is a directory /etc/leviathan_pass where each password can be found for each level. Of course you only have access to a level's password if you have the correct permissions

Leviathan 0 → 1:

Upon logging into level0 of leviathan, I was greeted with an empty directory. Great start. I just maneuvered around the directories a little and found that each level is contained within the home directory, and that at the root directory there are a bunch of folders such as a /tmp directory and /etc. So I went back to home and used the `find` command to see if there were any hidden files or directories. Sure enough there was a hidden directory called backup, with an html file called bookmarks.html, and I had no other leads so I decided to check it out. I shot the output of the file to stdout using `cat` and then piped that into `grep` searching for the pattern `leviathan1` which to my delight returned a series of strings which contained the leviathan1 password.

Leviathan 1 → 2:

Logging into level two there is an executable file called check. Running the file it asks for a password. The first time I was just messing with the password, it doesn't look like it had any sort of length limitation, so I found a command called `ltrace`. What `ltarce` intercepts and records the dynamic library calls which are called by the executed process and the signals which are received by that process. It can also intercept and print the system calls executed by the program. The part I was interested in was intercepting and printing system calls executed by the program. I made a disassembly of the executable file using the `objdump` command and saw that there were 3 getchars, and a string compare. What this told me was that there was a limit on how big the password could be (and that when I put more than that it was getting truncated) and that to do the comparison, it was using a function called strcmp (which stands for string compare). Since it was using a function to compare the strings, I knew that if I used ltrace, the strcmp

would be intercepted. So I ran the program by doing `echo test | ltrace ./check` and just put in a random password. Then ltrace presented what function calls the program check was making:

```
leviathan1@leviathan:~$ echo test| ltrace ./check
__libc_start_main(0x804853b, 1, 0xffffd784, 0x8048610 <unfinished ...>
printf("password: ")
getchar(1, 0, 0x65766f6c, 0x646f6700)
getchar(1, 0, 0x65766f6c, 0x646f6700)
getchar(1, 0, 0x65766f6c, 0x646f6700)
strcmp("tes", "sex")
puts("Wrong password, Good Bye ..."password: Wrong password, Good Bye ...
)
+++ exited (status 0) +++
```

The string compare tells us that the password the program is looking for is ‘sex’. So we can give the program this password, and the program then opens up a shell. I ran the command `id` to figure out if the shell gave me any special permissions, and sure enough my user id was labeled as leviathan2, so I just got the password from the /etc/leviathan_pass/ directory.

Leviathan 2 → 3:

This level required a ton of research. Upon logging into this level, we see there is an executable called `printfile` which when run with no arguments, shows us how to use it. It informs us it is a file printer and that with it you can print a file as long as you give it the path to the file. I figure this won’t work but lets try just running this with the path for the leviathan3 password:

```
leviathan2@leviathan:~$ ./printfile
*** File Printer ***
Usage: ./printfile filename
leviathan2@leviathan:~$ ./printfile /etc/leviathan_pass/leviathan3
You cant have that file...
```

As expected, it knows not to let us have files that typically don’t have access too. Lets run the binary with `ltrace` and see if it is making any important system calls:

```
leviathan2@leviathan:~$ ltrace ./printfile /etc/leviathan_pass/leviathan3
__libc_start_main(0x804852b, 2, 0xffffd764, 0x8048610 <unfinished ...>
access("/etc/leviathan_pass/leviathan3", 4)      = -1
puts("You cant have that file..."You cant have that file...
)                 = 27
+++ exited (status 1) +++
```

So it's running a c function called access which essentially checks to see if we have certain permissions to a file (the permissions are determined by the number code, in this case 4). After the access call, if we have the right permissions, it makes a system call to open the file:

```
leviathan2@leviathan:~$ ltrace ./printfile /tmp/boi123/test.txt
__libc_start_main(0x804852b, 2, 0xffffd774, 0x8048610 <unfinished ...>
access("/tmp/boi123/test.txt", 4)                  = 0
snprintf("/bin/cat /tmp/boi123/test.txt", 511, "/bin/cat %s", "/tmp/boi123/test.
txt") = 29
geteuid()                                          = 12002
geteuid()                                          = 12002
setreuid(12002, 12002)                             = 0
system("/bin/cat /tmp/boi123/test.txt" <no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )                             = 0
+++ exited (status 0) +++
```

There is a way to cheat the software however, and that is by using a file with spaces in the name that I have access to, as well as having a symbolic link to the file I want (https://en.wikipedia.org/wiki/Symbolic_link, and https://kb.iu.edu/d/abbe for more about symbolic links). First I wanted a file that I had access to, so that the access function would pass me through to the next steps of the program, the point of having a space in the filename is for when I get to the system call. If there is a space in the filename the call to system will recognize the file as two separate entities. For example, in the files I used: `symlink` and `symlink space` when I pass `symlink space` into access, it will say that yes I have access to the file, but when that get passed to system call, it will look at that and interpret `symlink` as the file we want to open, and space as an unrecognized command. Then, if the file we want to open is symlink, we can make the file `symlink` a symbolic link to the password for leviathan3:

```
leviathan2@leviathan:~$ cd /tmp/boi123
leviathan2@leviathan:/tmp/boi123$ ls
symlink  symlink space  test.txt
leviathan2@leviathan:/tmp/boi123$ file symlink
symlink: symbolic link to /etc/leviathan_pass/leviathan3
leviathan2@leviathan:/tmp/boi123$ cd
leviathan2@leviathan:~$ ./printfile /tmp/boi123/symlink\ space
Ahdiemoo1j
/bin/cat: space: No such file or directory
leviathan2@leviathan:~$
```

This will reveal the password.

Leviathan 3 → 4:

Upon entering the directory, I listed what was in that directory and found that there is a 32-bit LSB executable. I ran the executable, and tested it a couple of times with two different inputs, then I decided to run it with ltrace:

```
leviathan3@leviathan:~$ ls -al
total 32
drwxr-xr-x  2 root       root        4096 Aug 26  2019 .
drwxr-xr-x 10 root       root        4096 Aug 26  2019 ..
-rw-r--r--  1 root       root         220 May 15  2017 .bash_logout
-rw-r--r--  1 root       root        3526 May 15  2017 .bashrc
-r-sr-x---  1 leviathan4 leviathan3 10288 Aug 26  2019 level3
-rw-r--r--  1 root       root         675 May 15  2017 .profile
leviathan3@leviathan:~$ ./level3
Enter the password>
bzzzzzzzzap. WRONG
leviathan3@leviathan:~$ ./level3
Enter the password> hello worls
bzzzzzzzzap. WRONG
leviathan3@leviathan:~$ ltrace ./level3
__libc_start_main(0x8048618, 1, 0xffffd784, 0x80486d0 <unfinished ...>
strcmp("h0no33", "kakaka")                      = -1
printf("Enter the password> ")                  = 20
fgets(Enter the password> test
"test\n", 256, 0xf7fc55a0)                 = 0xffffd590
strcmp("test\n", "snlprintf\n")                 = 1
puts("bzzzzzzzzap. WRONG"bzzzzzzzzap. WRONG
)                       = 19
+++ exited (status 0) +++
leviathan3@leviathan:~$
```

The first string compare that happens between “h0no33” and “kakaka” I don’t really understand, but I do see that there is a strcmp with the input that I gave and a string “snlprintf\n” (the \n is just the representation of the new line character, aka hitting enter). So let's just try giving that string as input:

```
leviathan3@leviathan:~$ ./level3
Enter the password> snlprintf
[You've got shell]!
$
```

And just like that we have a shell! So I decided to run the id command to see who we are, and it turns out we are user leviathan4, so let's just open our password file:

```
[You've got shell]!
$ id
uid=12004(leviathan4) gid=12003(leviathan3) groups=12003(leviathan3)
$ cat /etc/leviathan_pass/leviathan4
```

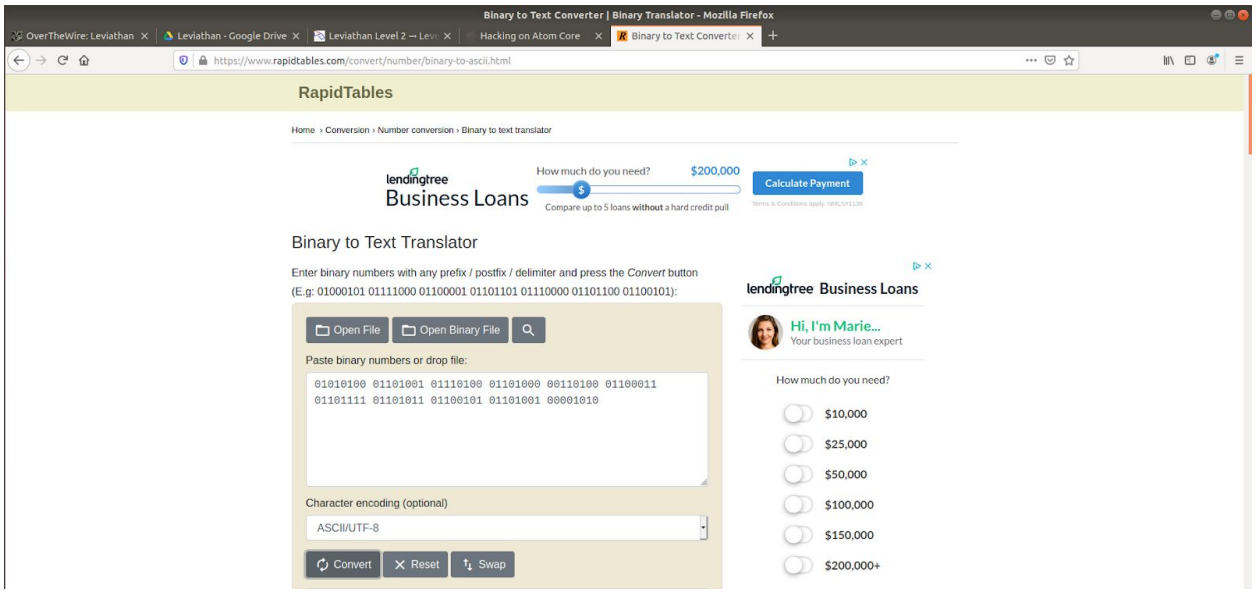
This will reveal the password.

Leviathan 4 → 5:

Upon loading into this level, I listed all of the contents of the home directory, and saw nothing, so I used ls -al to list everything and found a directory called `./.trash` which had a file called `bin`. Running bin prints some binary strings:

```
leviathan4@leviathan:~$ ls -la
total 24
drwxr-xr-x  3 root root       4096 Aug 26  2019 .
drwxr-xr-x 10 root root       4096 Aug 26  2019 ..
-rw-r--r--  1 root root        220 May 15  2017 .bash_logout
-rw-r--r--  1 root root       3526 May 15  2017 .bashrc
-rw-r--r--  1 root root        675 May 15  2017 .profile
dr-xr-x---  2 root leviathan4 4096 Aug 26  2019 .trash
leviathan4@leviathan:~$ cd ./.trash
leviathan4@leviathan:~/.trash$ ls
bin
leviathan4@leviathan:~/.trash$ ./bin
01010100 01101001 01110100 01101000 00110100 01100011 01101111 01101011 01100101
 01101001 00001010
leviathan4@leviathan:~/.trash$
```

Just a hunch, my thought process was that because all the binary strings are 8-bits I think they may be ascii codes. The ascii code assigns an 8-bit binary number to each symbol in the ascii alphabet. Instead of looking up a table and painstakingly translating every binary string, I just went online and looked up a converter and pasted in the binary input:



Doing this will reveal the password.

Leviathan 5 → 6:

Upon logging into this level, we found that there was an executable called leviathan5. I ran it once without ltrace and it showed that the program cannot find a certain file it is looking for:

```
leviathan5@leviathan:~$ find
.
./.bash_logout
./.profile
./.bashrc
./leviathan5
leviathan5@leviathan:~$ ./leviathan5
Cannot find /tmp/file.log
```

Assuming that the file it was looking for would be something that it would be using for an important process, I thought that what if we could make the file it is looking for, but link it to the password for leviathan 6. So I created a symbolic link with the file path that the program is looking for, and then ran the program again:

```
leviathan5@leviathan:~$ ln -s /etc/leviathan_pass/leviathan6 /tmp/file.log
leviathan5@leviathan:~$ ./leviathan5
```

This reveals the password.

Leviathan 6 → 7:

Upon logging into this level, we see that there is an executable called leviathan6, running it tells us how to use it:

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

Putting in the right code will hopefully return the passcode for the next level, so let's just brute force the pin code. I created a new directory in /tmp and wrote a small bash script that would generate all the possible four digit pin codes and input them into leviathan6:

```
!#/bin/bash

for i in {0000..9999}
        do
                echo /home/leviathan6/leviathan6 $i
        done
```

Now we need to run chmod on our script to make it executable:

```
leviathan6@leviathan:/tmp/test145$ ls
bruteforce.sh
leviathan6@leviathan:/tmp/test145$ chmod +x bruteforce.sh
leviathan6@leviathan:/tmp/test145$ ls
bruteforce.sh
leviathan6@leviathan:/tmp/test145$
```

And now we can run our script and see if it works:

```
Wrong
Wrong
Wrong
Wrong
Wrong
Wrong
Wrong
Wrong
Wrong
Wrong
Wrong
$ ls
bruteforce.sh
$ ^C
$ id
uid=12007(leviathan7) gid=12006(leviathan6) groups=12006(leviathan6)
$
```

And we can see above that eventually a shell opens up and we are greeted with the fact we are now using the machine as user leviathan7, so we can now just grab the password from the password directory.

We have now finished leviathan, if you'd like go into leviathan7 for a nice congratulatory message!