**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:**

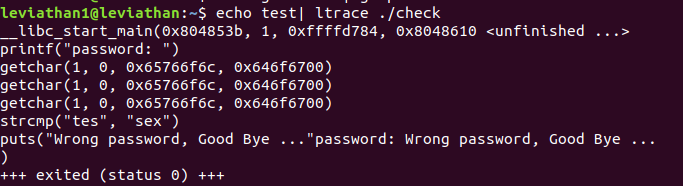
Just like in Bandit, there is a directory /etc/leviathan\_pass where each password can be found for each level. Of course you only have access to a level’s password if you have the correct permissions

**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:**

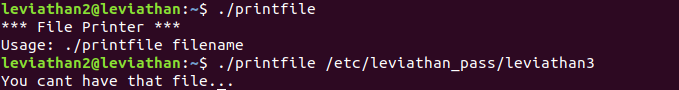
Logging into level two there is an executable file called check. Running the file it asks for a password. The first time I was just messing with the password, it doesn’t look like it had any sort of length limitation, so I found a command called ltrace. What ltarce intercepts and records the dynamic 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:



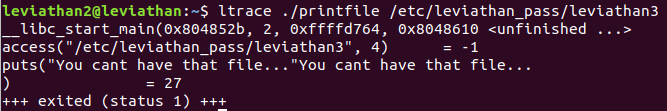
The string compare tells us that the password the program is looking for is ‘sex’. So we can give the program this password, and the program then opens up a shell. I ran the command id to figure out if the shell gave me any special permissions, and sure enough my user id was labeled as leviathan2, so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

**Leviathan 2 → 3:**

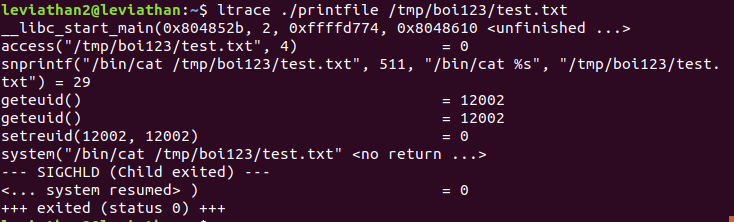
This level required a ton of research. Upon logging into this level, we see there is an executable called printfile which when run with no arguments, shows us how to use it. It informs us it is a file printer and that with it you can print a file as long as you give it the path to the file. I figure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



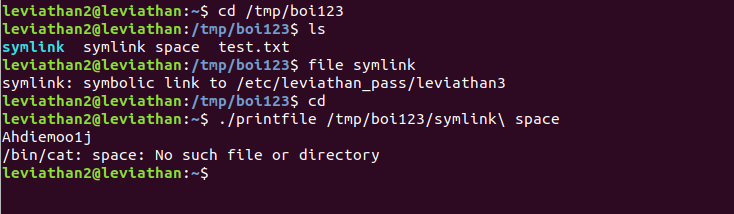
As expected, it knows not to let us have files that typically don’t have access too. Lets run the binary with ltrace and see if it is making any important system calls:



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:



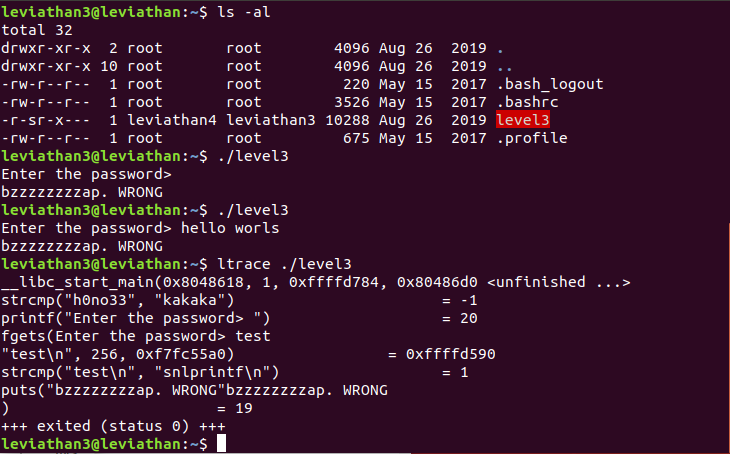
There is a way to cheat the software however, and that is by using a file with spaces in the name that I have access to, as well as having a symbolic link to the file I want (<https://en.wikipedia.org/wiki/Symbolic_link>, and <https://kb.iu.edu/d/abbe> for more about symbolic links). First I wanted a 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:



This will reveal the password.

**Leviathan 3 → 4:**

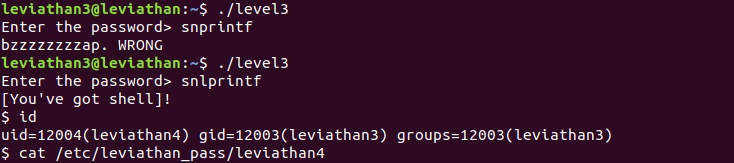
Upon entering the directory, I listed what was in that directory and found that there is a 32-bit LSB executable. I ran the executable, and tested it a couple of times with two different inputs, then I decided to run it with ltrace:



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:



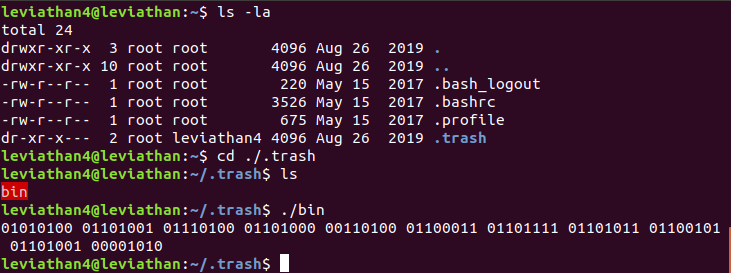
And just like that we have a shell! So I decided to run the id command to see who we are, and it turns out we are user leviathan4, so let's just open our password file:



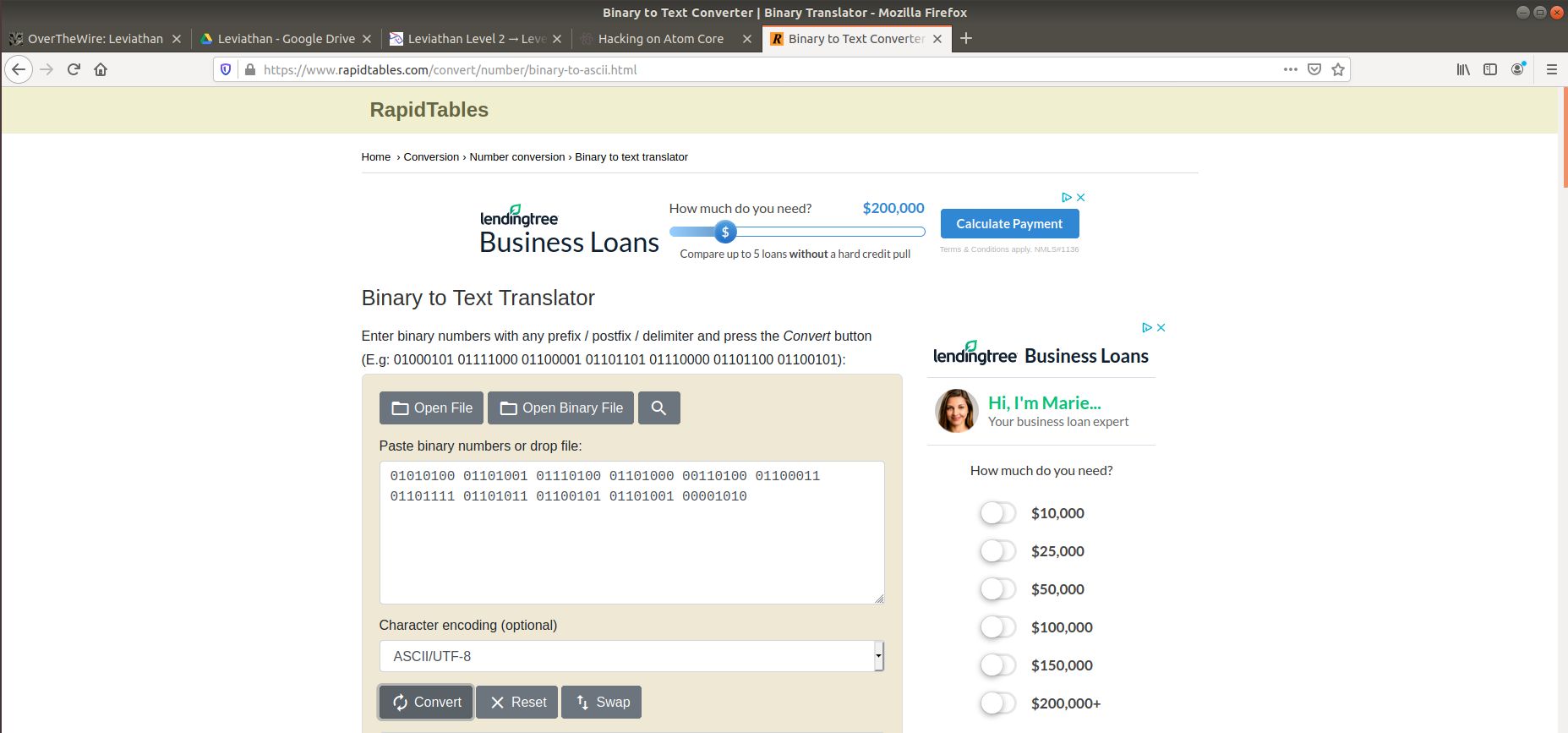
This will reveal the password.

**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:



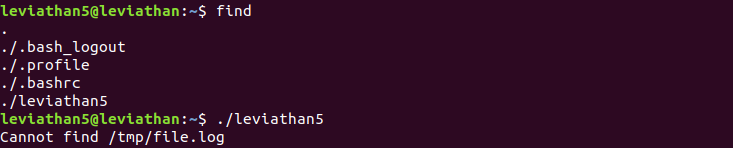
Just a hunch, my thought process was that because all the binary strings are 8-bits I think they may be ascii codes. The ascii code assigns an 8-bit binary number to each symbol in the ascii alphabet. Instead of looking up a table and painstakingly translating every binary string, I just went online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



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:

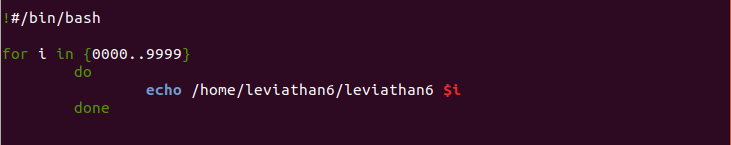


This reveals the password.

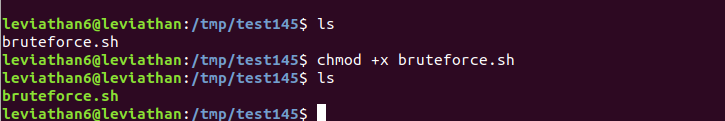
**Leviathan 6 → 7:**

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

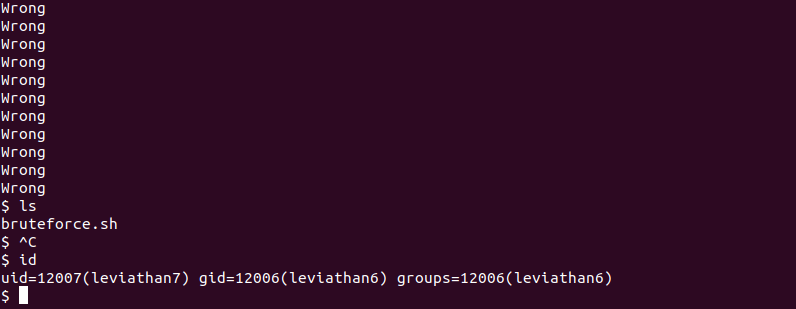
Putting in the right code will hopefully return the passcode for the next level, so let's just brute force the pin code. I created a new directory in /tmp and wrote a small bash script that would generate all the possible four digit pin codes and input them into leviathan6:



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



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



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!