djw67_1:

password (w/o quotes): "PRzhUWkjfrsTLmeLWgSmoauvjXM"

used strings and tried each string found in executable
script command used:

```
strings djw67_1 > out1.txt; while read p; do outTxt=`echo $p | ./djw67_1`; if [[ $outTxt == *"Congrat"* ]]; then echo $p; fi; done <out1.txt
```

djw67_2:

password (w/o quotes): "3.141593"

used gdb and stepped through the program. noticed that register mm7 was set to PI (3.141593....), which was very suspicious. Later found that a function 'd()' was called in main(), which then called c(), s(), and e() -- which all contained logic that eventually set register %eax to a nonzero value if the user input was "3.141593", which then jumped to a printf("") call that said the passcode was correct

djw67_3:

tried every string (like djw67_1) to no avail

with gdb, main() isn't found, so debugging initially wasn't working. Used the following command to find the entry point:

```
readelf -a djw67_3
```

which showed that the entry point is the '.text' label (w/o quotes).

setting a breakpoint at .text (break *0x080483a0) allowed me to debug from the entry point.

this is a very tricky one. First, it jumps to a main function called __libc_start_main, which then jumps to a function that is dynamically located in memory (this is the main() function, right?)

The first thing this main() function does is call a function at f7f37aab, which I found to be within the getchar() routine. This was simply setting the current stack pointer (esp) into ebx.

I ran out of time with this one, but I can see where it's going with this executable. Note the disassembly in the main() function below:

```
1:    0xf7e2d974 <+4>:     call   0xf7f37aab
2:    0xf7e2d979 <+9>:     add    $0x18d687,%ebx
3:    0xf7e2d97f <+15>:    sub    $0x5c,%esp
4:    0xf7e2d982 <+18>:    mov    0x7c(%esp),%esi
5:    0xf7e2d986 <+22>:    mov    0x84(%esp),%eax
6:    0xf7e2d98d <+29>:    mov    -0x74(%ebx),%edx
```

(1) is ass assigning ebx to be the stack pointer (esp). then, (2) is adding 0x18d687 to ebx. (3) is subtracting 0x5c from the stack pointer, (4) is is assigning the value REFERENCED by the stack pointer plus 0x7c into esi, (5) is assigning the value REFERENCED by the stack pointer plus 0x84 into eax, and (6) is assigning the value REFERENCED by ebx minus 0x74 into edx. This seems to set the begginning of the logic on what will be considered a phrase to unlock this program.