# w7_debug

## Debugging tools

### strace - system calls

`strace -o tracelog FILE` save output of strace to tracelog

### ltrace - library calls

### strings- extract printable text strings

- `a` all sections. (default)
- `d or -data` only initialized and loaded data sections

### getline

The `getline` function in C is used to read a line of input from a stream, such as `stdin` (standard input), into a dynamically allocated buffer. It is a safer alternative to functions like `gets` or `fgets` because it handles dynamic memory allocation and prevents buffer overflows.

---

#### Syntax

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

1. `lineptr` **(char ):****
   - A pointer to a `char *` (buffer) where the input will be stored.
   - If `lineptr` is `NULL`, `getline` will allocate memory automatically.
   - If `lineptr` is not `NULL`, it assumes the buffer already exists and resizes it if needed.
2. `n` **(size_t *):**
   - A pointer to a `size_t` variable that indicates the current size of the buffer pointed to by `lineptr`.
   - `getline` updates this value if it reallocates the buffer.
3. `stream` **(FILE *):**
   - The input stream to read from, such as `stdin` for standard input or a file pointer.

---

#### Return Value

- On Success:
  - Returns the number of characters read (including the newline `\n` but excluding the null terminator `\0`).
- On Failure:
  - Returns `1` if the end-of-file (EOF) is reached or an error occurs.
  - Sets `errno` to indicate the error.

### gdb

`objdump -d FILE` display disassembly information

`run` run program (with arguments)

`bt` backtrace by reading stack

`b` create breakpoints

`d` delete breakpoints

`c` continue after hitting breakpoints

`p [variable]` print variable

`disas` disassembly

`info` information on registers or variables

`si` step into

`ni` next instruction

`x` examine memory

| Command | Format |
|---|---|
| `x/s` | Display as a string (null-terminated). |
| `x/x` | Display as raw hexadecimal. |
| `x/d` | Display as a signed decimal. |
| `x/u` | Display as an unsigned decimal. |
| `x/t` | Display as binary. |
| `x/i` | Display as machine instructions (disassemble). |

## x86-64 assembly

registers:

`%rip` program counter

`%rax` Accumulator register, often used for storing return values or temporary data. (TEMP)

`%eax` Lower 32 bits of `%rax`. Contains the result of operations or comparisons.

`%al` Lower 8 bits of `%rax`. Used for byte-level operations (like comparing a single character).

`%rcx, %rdx, %rsi, %rdi` General-purpose registers for passing arguments or intermediate data. (ARG)

`%esi` Lower 32 bits of `%rsi`.

`%rbp` Base pointer, often used to reference local variables within the current stack frame. (LCL)

`%rsp` Stack pointer, points to the top of the stack (used for function calls and local storage). (SP)

| 64-bit Register | Lower 32 bits | Lower 16 bits | Lower 8 bits (High/Low) |
|---|---|---|---|
| `%rax` | `%eax` | `%ax` | `%al` (low), `%ah` (high) |
| `%rbx` | `%ebx` | `%bx` | `%bl` (low), `%bh` (high) |
| `%rcx` | `%ecx` | `%cx` | `%cl` (low), `%ch` (high) |
| `%rdx` | `%edx` | `%dx` | `%dl` (low), `%dh` (high) |
| `%rsi` | `%esi` | `%si` | `%sil` (low) |
| `%rdi` | `%edi` | `%di` | `%dil` (low) |
| `%rsp` | `%esp` | `%sp` | `%spl` (low) |
| `%rbp` | `%ebp` | `%bp` | `%bpl` (low) |
| `%r8` – `%r15` | `%r8d` – `%r15d` | `%r8w` – `%r15w` | `%r8b` – `%r15b` |

`(%rax)` indirect addressing.

`4(%rax)` indirect addressing with offset 4 bytes

`$0x42` immediate value (constant)

### 1. Zero-Argument Instructions

These instructions operate without explicit operands. They often work with implicit registers or affect processor flags.

| Instruction | Description |
|---|---|
| `ret` | Return from a function (pop address from stack). |
| `leave` | Restore the stack frame by resetting `%rbp` and `%rsp`. |
| `nop` | No operation (does nothing, often used for padding). |
| `pushf` | Push the processor flags register onto the stack. |
| `popf` | Pop the top of the stack into the processor flags register. |
| `clc` | Clear the carry flag (used in arithmetic). |
| `stc` | Set the carry flag. |
| `hlt` | Halt the CPU until the next interrupt. |

### 2. One-Argument Instructions

These instructions operate on a single operand, which can be a register, memory address, or immediate value. The operation is often implicit (e.g., incrementing or negating the operand).

| Instruction | Description |
|---|---|
| `callq <addr>` | Calls a function at `<addr>` (pushes return address onto the stack). |

| | |
|---|---|
| `push <src>` | Push the value in `<src>` onto the stack. |
| `pop <dst>` | Pop the top of the stack into `<dst>`. |
| `inc <dst>` | Increment the value in `<dst>` by 1. |
| `dec <dst>` | Decrement the value in `<dst>` by 1. |
| `neg <dst>` | Negate the value in `<dst>` (two's complement). |
| `not <dst>` | Perform a bitwise NOT on `<dst>` (flip all bits). |
| `jmp <addr>` | Unconditionally jump to `<addr>`. |
| `call <addr>` | Call a function at `<addr>` (push return address onto stack). |
| `test <dst>` | Perform a logical AND between `<dst>` and `<dst>`, setting flags but discarding the result. |
| `setcc <dst>` | Set the byte in `<dst>` based on a condition (`cc` specifies the condition, e.g., `sete` for equal). |

### 3. Two-Argument Instructions

These are the most common instructions in x86-64. They perform an operation on the **source** operand (`<src>`) and store the result in the **destination** operand (`<dst>`). The operands can be registers, memory addresses, or immediate values.

| Instruction | Description |
|---|---|
| `mov <src>, <dst>` | Move the value from `<src>` to `<dst>`. |
| `add <src>, <dst>` | Add `<src>` to `<dst>` and store the result in `<dst>`. |
| `sub <src>, <dst>` | Subtract `<src>` from `<dst>` and store the result in `<dst>`. |
| `cmp <src>, <dst>` | Compare `<src>` with `<dst>` (sets flags, no result stored). |
| `and <src>, <dst>` | Perform a bitwise AND between `<src>` and `<dst>`, storing the result in `<dst>`. |
| `or <src>, <dst>` | Perform a bitwise OR between `<src>` and `<dst>`, storing the result in `<dst>`. |
| `xor <src>, <dst>` | Perform a bitwise XOR between `<src>` and `<dst>`, storing the result in `<dst>`. |
| `lea <src>, <dst>` | Load the effective address of `<src>` into `<dst>` (used for pointer arithmetic). |
| `imul <src>, <dst>` | Multiply `<src>` with `<dst>` and store the result in `<dst>`. |
| `shr <src>, <dst>` | Shift `<dst>` right by `<src>` bits, filling with zeroes. |
| `shl <src>, <dst>` | Shift `<dst>` left by `<src>` bits, filling with zeroes. |
| `movzbl <src>, <dst>` | Move the zero-extended byte from `<src>` to `<dst>` (used to load 8-bit values into larger registers). |

## Cracks

```
$ strings -d ./crackme-1
...
strcmp
...
Beetlejuice
...

$ ltrace ./crackme-1
puts("What is the password?"What is the password?
)                                                = 22
getline(0x7fffbd391880, 0x7fffbd391878, 0x7f00de2449c0, 0x7fffbd391878
)   = 1
strcmp("", "Beetlejuice")                                                  = -66
puts("Nope"Nope
)                                                                 = 5
free(0x159a6b0)                                                            = <void>
+++ exited (status 1) +++
```

strcmp gives the password.

```
$ gdb ./crackme-2
(gdb) b main
(gdb) run <<< "abc"
(gdb) disas
Dump of assembler code for function main:
   0x0000000000400656 <+0>:     push   %rbp
```

```
   0x0000000000400657 <+1>:     mov    %rsp,%rbp
   0x000000000040065a <+4>:     sub    $0x20,%rsp
=> 0x000000000040065e <+8>:     movq   $0x0,-0x18(%rbp)
   0x0000000000400666 <+16>:    movq   $0x0,-0x20(%rbp)
   0x000000000040066e <+24>:    movq   $0x0,-0x10(%rbp)
   0x0000000000400676 <+32>:    mov    $0x400848,%edi
   0x000000000040067b <+37>:    callq  0x400550 <puts@plt>
   0x0000000000400680 <+42>:    mov    0x2009b9(%rip),%rdx        # 0x601040 <stdin@@GLIBC_2.2.5>
   0x0000000000400687 <+49>:    lea    -0x20(%rbp),%rcx
   0x000000000040068b <+53>:    lea    -0x18(%rbp),%rax
   0x000000000040068f <+57>:    mov    %rcx,%rsi
   0x0000000000400692 <+60>:    mov    %rax,%rdi
   0x0000000000400695 <+63>:    callq  0x400560 <getline@plt>
   0x000000000040069a <+68>:    mov    %rax,-0x10(%rbp)
   0x000000000040069e <+72>:    mov    -0x18(%rbp),%rax
   0x00000000004006a2 <+76>:    mov    -0x10(%rbp),%rdx
   0x00000000004006a6 <+80>:    sub    $0x1,%rdx
   0x00000000004006aa <+84>:    add    %rdx,%rax
   0x00000000004006ad <+87>:    movb   $0x0,(%rax)
   0x00000000004006b0 <+90>:    cmpq   $0xb,-0x10(%rbp)
   0x00000000004006b5 <+95>:    jne    0x40077b <main+293>
   0x00000000004006bb <+101>:   mov    -0x18(%rbp),%rax
   0x00000000004006bf <+105>:   movzbl (%rax),%eax
   0x00000000004006c2 <+108>:   cmp    $0x42,%al
   0x00000000004006c4 <+110>:   jne    0x40077b <main+293>
   0x00000000004006ca <+116>:   mov    -0x18(%rbp),%rax
   0x00000000004006ce <+120>:   add    $0x2,%rax
   0x00000000004006d2 <+124>:   movzbl (%rax),%eax
   0x00000000004006d5 <+127>:   cmp    $0x74,%al
   0x00000000004006d7 <+129>:   jne    0x40077b <main+293>
   0x00000000004006dd <+135>:   mov    -0x18(%rbp),%rax
   0x00000000004006e1 <+139>:   add    $0x8,%rax
   0x00000000004006e5 <+143>:   movzbl (%rax),%eax
   0x00000000004006e8 <+146>:   cmp    $0x73,%al
   0x00000000004006ea <+148>:   jne    0x40077b <main+293>
   0x00000000004006f0 <+154>:   mov    -0x18(%rbp),%rax
   0x00000000004006f4 <+158>:   add    $0x3,%rax
   0x00000000004006f8 <+162>:   movzbl (%rax),%eax
   0x00000000004006fb <+165>:   cmp    $0x65,%al
   0x00000000004006fd <+167>:   jne    0x40077b <main+293>
   0x00000000004006ff <+169>:   mov    -0x18(%rbp),%rax
   0x0000000000400703 <+173>:   add    $0x9,%rax
   0x0000000000400707 <+177>:   movzbl (%rax),%eax
   0x000000000040070a <+180>:   cmp    $0x65,%al
   0x000000000040070c <+182>:   jne    0x40077b <main+293>
   0x000000000040070e <+184>:   mov    -0x18(%rbp),%rax
   0x0000000000400712 <+188>:   add    $0x6,%rax
   0x0000000000400716 <+192>:   movzbl (%rax),%eax
   0x0000000000400719 <+195>:   cmp    $0x65,%al
   0x000000000040071b <+197>:   jne    0x40077b <main+293>
   0x000000000040071d <+199>:   mov    -0x18(%rbp),%rax
   0x0000000000400721 <+203>:   add    $0x1,%rax
   0x0000000000400725 <+207>:   movzbl (%rax),%eax
   0x0000000000400728 <+210>:   cmp    $0x65,%al
   0x000000000040072a <+212>:   jne    0x40077b <main+293>
   0x000000000040072c <+214>:   mov    -0x18(%rbp),%rax
   0x0000000000400730 <+218>:   add    $0x5,%rax
   0x0000000000400734 <+222>:   movzbl (%rax),%eax
   0x0000000000400737 <+225>:   cmp    $0x67,%al
   0x0000000000400739 <+227>:   jne    0x40077b <main+293>
   0x000000000040073b <+229>:   mov    -0x18(%rbp),%rax
   0x000000000040073f <+233>:   add    $0x4,%rax
   0x0000000000400743 <+237>:   movzbl (%rax),%eax
   0x0000000000400746 <+240>:   cmp    $0x6c,%al
```

```
   0x0000000000400748 <+242>:   jne    0x40077b <main+293>
   0x000000000040074a <+244>:   mov    -0x18(%rbp),%rax
   0x000000000040074e <+248>:   add    $0x7,%rax
   0x0000000000400752 <+252>:   movzbl (%rax),%eax
   0x0000000000400755 <+255>:   cmp    $0x75,%al
   0x0000000000400757 <+257>:   jne    0x40077b <main+293>
   0x0000000000400759 <+259>:   mov    -0x18(%rbp),%rax
   0x000000000040075d <+263>:   add    $0xa,%rax
   0x0000000000400761 <+267>:   movzbl (%rax),%eax
   0x0000000000400764 <+270>:   test   %al,%al
   0x0000000000400766 <+272>:   jne    0x40077b <main+293>
   0x0000000000400768 <+274>:   mov    $0x40085e,%edi
   0x000000000040076d <+279>:   callq  0x400550 <puts@plt>
   0x0000000000400772 <+284>:   movl   $0x0,-0x4(%rbp)
   0x0000000000400779 <+291>:   jmp    0x40078c <main+310>
   0x000000000040077b <+293>:   mov    $0x400867,%edi
   0x0000000000400780 <+298>:   callq  0x400550 <puts@plt>
   0x0000000000400785 <+303>:   movl   $0x1,-0x4(%rbp)
   0x000000000040078c <+310>:   mov    -0x18(%rbp),%rax
   0x0000000000400790 <+314>:   test   %rax,%rax
   0x0000000000400793 <+317>:   je     0x4007a1 <main+331>
   0x0000000000400795 <+319>:   mov    -0x18(%rbp),%rax
   0x0000000000400799 <+323>:   mov    %rax,%rdi
   0x000000000040079c <+326>:   callq  0x400540 <free@plt>
   0x00000000004007a1 <+331>:   mov    -0x4(%rbp),%eax
   0x00000000004007a4 <+334>:   leaveq
   0x00000000004007a5 <+335>:   retq
End of assembler dump.
```

look up all the *cmp*s:

```
$ objdump -d ./crackme-2 | grep -w cmp
  4005be:       48 39 f8                cmp    %rdi,%rax
  4006c2:       3c 42                   cmp    $0x42,%al
  4006d5:       3c 74                   cmp    $0x74,%al
  4006e8:       3c 73                   cmp    $0x73,%al
  4006fb:       3c 65                   cmp    $0x65,%al
  40070a:       3c 65                   cmp    $0x65,%al
  400719:       3c 65                   cmp    $0x65,%al
  400728:       3c 65                   cmp    $0x65,%al
  400737:       3c 67                   cmp    $0x67,%al
  400746:       3c 6c                   cmp    $0x6c,%al
  400755:       3c 75                   cmp    $0x75,%al
  400801:       48 39 dd                cmp    %rbx,%rbp
```

the first arguments form the password in order.

```
$ gdb ./crackme-3
(gdb) b main
(gdb) run <<< "abc"
(gdb) disas
Dump of assembler code for function main:
   0x00000000004006f6 <+0>:     push   %rbp
   0x00000000004006f7 <+1>:     mov    %rsp,%rbp
   0x00000000004006fa <+4>:     sub    $0x20,%rsp
=> 0x00000000004006fe <+8>:     movq   $0x0,-0x18(%rbp)
   0x0000000000400706 <+16>:    movq   $0x0,-0x20(%rbp)
   0x000000000040070e <+24>:    movl   $0x0,-0xc(%rbp)
   0x0000000000400715 <+31>:    mov    $0x400898,%edi
   0x000000000040071a <+36>:    callq  0x4005d0 <puts@plt>
   0x000000000040071f <+41>:    mov    0x20092a(%rip),%rdx        # 0x601050 <stdin@@GLIBC_2.2.5>
   0x0000000000400726 <+48>:    lea    -0x20(%rbp),%rcx
   0x000000000040072a <+52>:    lea    -0x18(%rbp),%rax
   0x000000000040072e <+56>:    mov    %rcx,%rsi
```

```
   0x0000000000400731 <+59>:    mov    %rax,%rdi
   0x0000000000400734 <+62>:    callq  0x400600 <getline@plt>
   0x0000000000400739 <+67>:    mov    %eax,-0xc(%rbp)
   0x000000000040073c <+70>:    cmpl   $0x0,-0xc(%rbp)
   0x0000000000400740 <+74>:    jns    0x40074c <main+86>
   0x0000000000400742 <+76>:    mov    $0x1,%eax
   0x0000000000400747 <+81>:    jmpq   0x4007fd <main+263>
   0x000000000040074c <+86>:    mov    -0x18(%rbp),%rax
   0x0000000000400750 <+90>:    mov    -0xc(%rbp),%edx
   0x0000000000400753 <+93>:    movslq %edx,%rdx
   0x0000000000400756 <+96>:    sub    $0x1,%rdx
   0x000000000040075a <+100>:   add    %rdx,%rax
   0x000000000040075d <+103>:   movb   $0x0,(%rax)
   0x0000000000400760 <+106>:   movl   $0x0,-0x8(%rbp)
   0x0000000000400767 <+113>:   jmp    0x40078f <main+153>
   0x0000000000400769 <+115>:   mov    -0x18(%rbp),%rdx
   0x000000000040076d <+119>:   mov    -0x8(%rbp),%eax
   0x0000000000400770 <+122>:   cltq
   0x0000000000400772 <+124>:   add    %rdx,%rax
   0x0000000000400775 <+127>:   movzbl (%rax),%ecx
   0x0000000000400778 <+130>:   mov    -0x18(%rbp),%rdx
   0x000000000040077c <+134>:   mov    -0x8(%rbp),%eax
   0x000000000040077f <+137>:   cltq
   0x0000000000400781 <+139>:   add    %rdx,%rax
   0x0000000000400784 <+142>:   xor    $0x42,%ecx
   0x0000000000400787 <+145>:   mov    %ecx,%edx
   0x0000000000400789 <+147>:   mov    %dl,(%rax)
   0x000000000040078b <+149>:   addl   $0x1,-0x8(%rbp)
   0x000000000040078f <+153>:   mov    -0xc(%rbp),%eax
   0x0000000000400792 <+156>:   sub    $0x1,%eax
   0x0000000000400795 <+159>:   cmp    %eax,-0x8(%rbp)
   0x0000000000400798 <+162>:   jl     0x400769 <main+115>
   0x000000000040079a <+164>:   mov    -0x18(%rbp),%rax
   0x000000000040079e <+168>:   mov    %rax,%rdi
   0x00000000004007a1 <+171>:   callq  0x4005e0 <strlen@plt>
   0x00000000004007a6 <+176>:   cmp    $0xa,%rax
   0x00000000004007aa <+180>:   jne    0x4007d4 <main+222>
   0x00000000004007ac <+182>:   mov    -0x18(%rbp),%rax
   0x00000000004007b0 <+186>:   mov    $0x4008ae,%esi
   0x00000000004007b5 <+191>:   mov    %rax,%rdi
   0x00000000004007b8 <+194>:   callq  0x4005f0 <strcmp@plt>
   0x00000000004007bd <+199>:   test   %eax,%eax
   0x00000000004007bf <+201>:   jne    0x4007d4 <main+222>
   0x00000000004007c1 <+203>:   mov    $0x4008b9,%edi
   0x00000000004007c6 <+208>:   callq  0x4005d0 <puts@plt>
   0x00000000004007cb <+213>:   movl   $0x0,-0x4(%rbp)
   0x00000000004007d2 <+220>:   jmp    0x4007e5 <main+239>
   0x00000000004007d4 <+222>:   mov    $0x4008c2,%edi
   0x00000000004007d9 <+227>:   callq  0x4005d0 <puts@plt>
   0x00000000004007de <+232>:   movl   $0x1,-0x4(%rbp)
   0x00000000004007e5 <+239>:   mov    -0x18(%rbp),%rax
   0x00000000004007e9 <+243>:   test   %rax,%rax
   0x00000000004007ec <+246>:   je     0x4007fa <main+260>
   0x00000000004007ee <+248>:   mov    -0x18(%rbp),%rax
   0x00000000004007f2 <+252>:   mov    %rax,%rdi
   0x00000000004007f5 <+255>:   callq  0x4005c0 <free@plt>
   0x00000000004007fa <+260>:   mov    -0x4(%rbp),%eax
   0x00000000004007fd <+263>:   leaveq
   0x00000000004007fe <+264>:   retq
End of assembler dump.
```

from `main+115` to `main+162` is a loop iterating through the input string and XORs each character with `0x42`. strcmp is called at main+194 callq (0x4005f0 is where strcmp is stored at). strcmp always compares `%rsi` and `%rdi`, so we look up `main+191` and `main+186`, and found an address 0x4008ae. type `x/s 0x4008ae` returns

```
(gdb) x/s 0x4008ae
0x4008ae:       "\017'\021#;\006#;r*"
```

which contains escaped octal code (since they are non-printable characters). revert this string with xor 0x42.

```
$ ltrace ./crackme-4
puts("What is the password?"What is the password?
)                                               = 22
getline(0x7ffc7d25b0c8, 0x7ffc7d25b0c0, 0x7f9c26e169c0, 0x7ffc7d25b0c0
)   = 1
srand(0)                                                                  = <void>
rand()                                                                    = 1804289383
atoi(0xe7a6b0, 0x7ffc7d25b094, 0, 0x7f9c26e161e8)                         = 0
puts("Nope"Nope
)                                                               = 5
free(0xe7a6b0)                                                            = <void>
+++ exited (status 1) +++

$ gdb ./crackme-4
(gdb) b main
(gdb) run <<< "password"
(gdb) disas
Dump of assembler code for function main:
   0x0000000000400736 <+0>:     push   %rbp
   0x0000000000400737 <+1>:     mov    %rsp,%rbp
   0x000000000040073a <+4>:     sub    $0x20,%rsp
=> 0x000000000040073e <+8>:     movq   $0x0,-0x18(%rbp)
   0x0000000000400746 <+16>:    movq   $0x0,-0x20(%rbp)
   0x000000000040074e <+24>:    movq   $0x0,-0x10(%rbp)
   0x0000000000400756 <+32>:    mov    $0x400898,%edi
   0x000000000040075b <+37>:    callq  0x400600 <puts@plt>
   0x0000000000400760 <+42>:    mov    0x2008e9(%rip),%rdx        # 0x601050 <stdin@@GLIBC_2.2.5>
   0x0000000000400767 <+49>:    lea    -0x20(%rbp),%rcx
   0x000000000040076b <+53>:    lea    -0x18(%rbp),%rax
   0x000000000040076f <+57>:    mov    %rcx,%rsi
   0x0000000000400772 <+60>:    mov    %rax,%rdi
   0x0000000000400775 <+63>:    callq  0x400630 <getline@plt>
   0x000000000040077a <+68>:    mov    %rax,-0x10(%rbp)
   0x000000000040077e <+72>:    mov    -0x18(%rbp),%rax
   0x0000000000400782 <+76>:    mov    -0x10(%rbp),%rdx
   0x0000000000400786 <+80>:    sub    $0x1,%rdx
   0x000000000040078a <+84>:    add    %rdx,%rax
   0x000000000040078d <+87>:    movb   $0x0,(%rax)
   0x0000000000400790 <+90>:    mov    $0x0,%edi
   0x0000000000400795 <+95>:    callq  0x400610 <srand@plt>
   0x000000000040079a <+100>:   callq  0x400640 <rand@plt>
   0x000000000040079f <+105>:   mov    %eax,-0x4(%rbp)
   0x00000000004007a2 <+108>:   mov    -0x18(%rbp),%rax
   0x00000000004007a6 <+112>:   mov    %rax,%rdi
   0x00000000004007a9 <+115>:   callq  0x400620 <atoi@plt>
   0x00000000004007ae <+120>:   cmp    %eax,-0x4(%rbp)
   0x00000000004007b1 <+123>:   jne    0x4007c6 <main+144>
   0x00000000004007b3 <+125>:   mov    $0x4008ae,%edi
   0x00000000004007b8 <+130>:   callq  0x400600 <puts@plt>
   0x00000000004007bd <+135>:   movl   $0x0,-0x4(%rbp)
   0x00000000004007c4 <+142>:   jmp    0x4007d7 <main+161>
   0x00000000004007c6 <+144>:   mov    $0x4008b7,%edi
   0x00000000004007cb <+149>:   callq  0x400600 <puts@plt>
   0x00000000004007d0 <+154>:   movl   $0x1,-0x4(%rbp)
   0x00000000004007d7 <+161>:   mov    -0x18(%rbp),%rax
   0x00000000004007db <+165>:   test   %rax,%rax
   0x00000000004007de <+168>:   je     0x4007ec <main+182>
   0x00000000004007e0 <+170>:   mov    -0x18(%rbp),%rax
   0x00000000004007e4 <+174>:   mov    %rax,%rdi
```

```
   0x00000000004007e7 <+177>:   callq  0x4005f0 <free@plt>
   0x00000000004007ec <+182>:   mov    -0x4(%rbp),%eax
   0x00000000004007ef <+185>:   leaveq
   0x00000000004007f0 <+186>:   retq
End of assembler dump.
```

combining `ltrace` with `gdb` we know at `main+95` the seed is set to `srand(0)`. there is only one `cmp`, namely `main+120`, and its second argument is `-0x4(%rbp)`. so we go all the back to find this address until `main+105`, where the value of `%eax` is sent to this address. and right above it is `callq` `rand`, so the `cmp` compares `%eax` from `main+115`, `callq` `atoi` with `rand`. and from `ltrace` we know the output of `rand` is `1804289383`. Alternative approach: extract the value of that address:

```
(gdb) b *0x4007ae
(gdb) run <<< "a"
(gdb) info registers rbp
rbp            0x7fffffffd3b0      0x7fffffffd3b0
(gdb) x/wx 0x7fffffffd3ac
0x7fffffffd3ac: 0x6b8b4567
(gdb) x/u 0x7fffffffd3ac
0x7fffffffd3ac: 1804289383
(gdb) x/d 0x7fffffffd3ac
0x7fffffffd3ac: 1804289383
```

also reveals the password.

```
$ strings -d ./crackme-5
...
strncmp
puts
tolower
getchar
toupper
...
B3t3Lg3uS3
...

$ ltrace ./crackme-5
puts("What is the password?"What is the password?
)                                               = 22
getchar(0, 0x25472a0, 0x7f0c39975860, 0x7f0c396d15a8
)                     = 10
strncmp("", "B3t3Lg3uS3", 10)                                                 = -66
puts("Nope"Nope
)                                                                 = 5
+++ exited (status 0) +++

$ strace ./crackme-5
execve("./crackme-5", ["./crackme-5"], 0x7fffc29fb9a0 /* 49 vars */) = 0
...
write(1, "What is the password?\n", 22What is the password?
) = 22
...
read(0,
"\n", 1024)                     = 1
write(1, "Nope\n", 5Nope
)                   = 5
exit_group(0)                           = ?
+++ exited with 0 +++

$ gdb ./crackme-5
(gdb) b main
(gdb) run <<< ""
(gdb) disas
Dump of assembler code for function main:
   0x0000000000400746 <+0>:     push   %rbp
```

```
   0x0000000000400747 <+1>:     mov    %rsp,%rbp
   0x000000000040074a <+4>:     sub    $0x10,%rsp
=> 0x000000000040074e <+8>:     mov    $0x400a78,%edi
   0x0000000000400753 <+13>:    callq  0x400620 <puts@plt>
   0x0000000000400758 <+18>:    jmp    0x400765 <main+31>
   0x000000000040075a <+20>:    movsbl -0x1(%rbp),%eax
   0x000000000040075e <+24>:    mov    %eax,%edi
   0x0000000000400760 <+26>:    callq  0x4007c1 <interpret>
   0x0000000000400765 <+31>:    callq  0x400630 <getchar@plt>
   0x000000000040076a <+36>:    mov    %al,-0x1(%rbp)
   0x000000000040076d <+39>:    cmpb   $0xff,-0x1(%rbp)
   0x0000000000400771 <+43>:    je     0x400779 <main+51>
   0x0000000000400773 <+45>:    cmpb   $0xa,-0x1(%rbp)
   0x0000000000400777 <+49>:    jne    0x40075a <main+20>
   0x0000000000400779 <+51>:    mov    $0xa,%edx
   0x000000000040077e <+56>:    mov    $0x400a8e,%esi
   0x0000000000400783 <+61>:    mov    $0x602070,%edi
   0x0000000000400788 <+66>:    callq  0x400600 <strncmp@plt>
   0x000000000040078d <+71>:    test   %eax,%eax
   0x000000000040078f <+73>:    jne    0x4007ab <main+101>
   0x0000000000400791 <+75>:    movzbl 0x2018e8(%rip),%eax        # 0x602080 <failed>
   0x0000000000400798 <+82>:    xor    $0x1,%eax
   0x000000000040079b <+85>:    test   %al,%al
   0x000000000040079d <+87>:    je     0x4007ab <main+101>
   0x000000000040079f <+89>:    mov    $0x400a99,%edi
   0x00000000004007a4 <+94>:    callq  0x400620 <puts@plt>
   0x00000000004007a9 <+99>:    jmp    0x4007b5 <main+111>
   0x00000000004007ab <+101>:   mov    $0x400aa2,%edi
   0x00000000004007b0 <+106>:   callq  0x400620 <puts@plt>
   0x00000000004007b5 <+111>:   movzbl 0x2018c4(%rip),%eax        # 0x602080 <failed>
   0x00000000004007bc <+118>:   movzbl %al,%eax
   0x00000000004007bf <+121>:   leaveq
   0x00000000004007c0 <+122>:   retq
End of assembler dump.
```

from disas we know there is an `interpreter` function altering the input string. we can use the following to determine which characters are strange:

```
$ ltrace ./crackme-5 <<< "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
$ ltrace ./crackme-5 <<< "abcdefghijklmnopqrstuvwxyz"
$ ltrace ./crackme-5 <<< "\!\@\#\$\%\^\&\*\(\)\-\_\+\=\{\}\[\]\|\\\:\;\"\'\<\>\,\.\?\/"
$ ltrace ./crackme-5 <<< "1234567890"
```

and can directly observe from the output that `c, d, l, U, +, -, >, <, .` are distinct and likely to correspond to commands. next step is go back to `gdb` and repeat the following process:

1. set breakpoint at interpret then run with the distinct chars.

   ```
   b interpret
   run <<< "++c>>d--l<<U."
   ```

2. use `si` to step next, use `p i` to observe the value of loop variable `i`, use `p command` to remind the current char being examined (current char of the input string), use `x/s memory` view the current value of memory, in order to deduce the utility of that char.
3. after figuring out the effect of every char, the puzzle is solved.